НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій, СТІРЕНКО

«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Сервіс для відкладеної відправки поштових повідомлень»

Виконав:
студент IV курсу, групи ІО-64
Андрійчук Дмитро Анатолійович _____

Керівник:
Асистент
Каплунов Артем Володимирович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
ст. викладач кафедри АУТС,
Яланецький Валерій Анатолійович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій, СТІРЕНКО
«___»_____2020 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Андрійчуку Дмитру Анатолійовичу

1. Тема проєкту «Сервіс для відкладеної відправки поштових повідомлень», керівник проєкту Каплунов Артем Володимирович, асистент, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту *26 травня 2020р.*

3. Вихідні дані до проєкту *див. технічне завдання*

4. Зміст пояснювальної записки *Аналіз і характеристика існуючих рішень об'єкта проектування, обґрунтування оптимального реалізації розроблюваного додатку, вибір інструментів та технологій і обґрунтування вибору, розробка додатку, основні рішення з реалізації. Висновки.*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) *принципова схема, функціональна схема, структурна схема*

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Сімоненко В. П., професор, д.т.н.		

7. Дата видачі завдання 01.09.2019

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	*Затвердження теми роботи*	*01.09.2019*	
2.	*Вивчення та аналіз завдання*	*15.12.2019-15.03.2020*	
3.	*Розробка архітектури додатку*	*15.03.2020-25.03.2020*	
4.	*Написання програмної частини*	*25.03.2020-05.04.2020*	
5.	*Тестування та виправлення помилок*	*05.04.2020-15.04.2020*	
6.	*Оформлення пояснювальної записки*	*15.04.2020-20.05.2020*	
7.	*Захист програмного продукту*	*25.04.2020*	
8.	*Передзахист*	*26.05.2020*	
9.	*Захист*	*18.06.2020*	

Студент Дмитро АНДРІЙЧУК

Керівник Артем КАПЛУНОВ

Анотація

В даному бакалаврському дипломному проекті розроблюється сервіс для відкладеної відправки поштових повідомлень. Не дивлячись на стрімкий розвиток інших технологій обміну повідомленнями, електронна пошта як і раніше найпоширеніша форма бізнес-спілкування.

Реалізований сервіс вирішує функціонал масової розсилки повідомлень у вказаний користувачем момент, не прибігаючи до використання існуючиш поштових сервісів до моменту відправки повідомлення, тим самим зберігаючи таємницю ще ненадісланих повідомлень від зламників, або працівників компанії, що володіє поштовим сервісом.

Запропонований додаток може цілком бути використаний при розробці корпоративної пошти компанії, адже написаний у добре масштабованому та придатному до розширення вигляді.

Аннотация

В данном бакалаврском дипломном проекте разрабатывается сервис для отложенной отправки почтовых сообщений. Несмотря на стремительное развиток других технологий обмена сообщениями, электронная почта по-прежнему самая распространенная форма бизнес-общения.

Реализован сервис решает функционал массовой рассылки сообщений в указанный пользователем момент, не прибегая к использованию существующих почтовых сервисов до момента отправки сообщения, тем самым сохраняя тайну еще не отправленных сообщений от взломщиков, или работников компании, владеющей почтовым сервисом.

Предложенный приложение может полностью быть использован при разработке корпоративной почты компании, ведь написано в хорошо масштабируемом и пригодном к расширению виде.

Abstract

In this bachelor's diploma project the Service for Delayed email Sending is developed. Despite the rapid development of other messaging technologies, e-mail is still the most common form of business communication.

The implemented service solves the functionality of mass mailing at the time specified by the user, without resorting to the use of existing e-mail services until the message is sent, thus keeping secret unsent messages from hackers or employees of the company that owns the e-mail service.

The proposed application can be fully used in the development of corporate mail of the company because it is written in a well-scalable and expandable form.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ДП 6401. 02.000 ТЗ	Технічне Завдання	3	
3	А4	ДП 6401. 03.000 ПЗ	Пояснювальна записка	62	
4	А4	ДП 6401. 04.000 Д1	Принципова схема	1	
5	А4	ДП 6401. 05.000 Д2	Функціональна схема	1	
6	А4	ДП 6401. 06.000 Д3	Структурна схема	1	
7	А4	ДП 6401. 07.000 Д4	Лістинг програми	19	

					ДП.6401.01.000 ВП		
Зм.		*№ документа*	*Підп.*	*Дата*			
Розробив		*Анлрійчук Д.А.*			*Сервіс для відкладеної відправки поштових повідомлень*	*Літ.*	*Аркуш*
Перевір.		*Каплунов А. В.*				Т	1
Н.конт		*Симоненко В.П.*			*Відомість дипломного проєкту*	*НТУУ «КПІ імені Ігора Сікорського», ФІОТ Група ІО - 64*	
Затв.							

Технічне завдання
до дипломного проєкту

на тему «Сервіс для відкладеної відправки поштових повідомлень»

ЗМІСТ

1.НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: «Сервіс для відкладеної відправки поштових повідомлень».

Область застосування: програма може бути використана для розсилки поштових повідомлень.

2.ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3.МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення сервісу для відкладеної відправки поштових повідомлень

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література, публікації в спеціалізованих періодичних виданнях, довідники по платформах дистанційного навчання, публікації в мережі Інтернет по даній темі.

5.ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

Додаток, що розробляється повинен:

- Надавати можливість авторизації та реэстрації користувачів
- Надавати можливість створювати, редагувати, переглядати, та відсилати повідомлення.
- Мати програмний інтерфейс
- Мати інтерфейс адміністратора

- Підключатись до існуючої БД

5.2. Вимоги до пристрою

- Компьютер на базі процессора Intel GTX серії 950 и вище
- Мінімальний обсяг оперативної пам'яті - 2 ГБ;
- Свободное пространство жесткого диска не менее 2 Гб
- Підключення до інтернету
- Наявність встановленої мови Python, СУБД PostgresQL, бібліотеки Django та модулів список яких надано у файлі requirements.txt

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення необхідної літератури	19.02.2020
Складання і узгодження технічного завдання	06.03.2020
Написання вступної частини та огляд рішень	19.03.2020
Розробка архітектури додатку	03.04.2020
Написання програмної частини	10.04.2020
Тестування та виправлення помилок	01.05.2020
Оформлення документації дипломного проекту	15.05.2020
Попередній захист та проходження нормативного контролю	29.05.2020
Захист дипломного проекту	15.06.2020

Пояснювальна записка
до дипломного проєкту

на тему: «Сервіс для відкладеної відправки поштових повідомлень»

Київ – 2020 року

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

БД	База даних
IMAP	Internet Message Access Protocol
ПЗ	Програмне забезпечення
LAN	Local area network
VPN	virtual private network
DNS	Domain Name System
SMTP	Simple Mail Transfer Protocol
IM	Instant Messaging
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
ORM	Object-relational mapping
MVC	Model–view–controller
MTV	Model–template–view
ASCII	American Standard Code for Information Interchange
ASGI	Asynchronous Server Gateway Interface
WSGI	Web Server Gateway Interface
UUID	Universally unique identifier

ВСТУП

В нашу сучасну добу майже в кожного користувача інтернету є власна електронна пошта і багато компаній надають доступ до сервісів обміну повідомленнями абсолютно безкоштовно, однак, у 2013 році Едвард Сноуден підтвердив те, що уряди держав стежать за вами. Спецслужби моніторять ваш трафік і створюють цифрові профілі ваших мережевих звичок. На жаль, вони не єдині, кому цікаві ваші дані. Хакери і кіберзлочинці також не проти перехопити вашу персональну інформацію.

Електронна пошта для більшості компаній є основним засобом комунікації. Переписка необхідна як для спілкування всередині самої компанії, так і для обміну даними з партнерами, клієнтами, постачальниками, державними органами і т.д. Ні для кого не секрет, що за корпоративною електронною поштою пересилається маса конфіденційної інформації: договори, рахунки, інформація про продукцію та ціни компанії, фінансові показники і т.д.

Якщо така інформація потрапляє в руки конкурентів, вона може суттєво зашкодити компанії аж до припинення її існування. Конкуренти, маючи в розпорядженні базу клієнтів, знаючи умови роботи і ціни, зможуть запропонувати їм кращі умови і, таким чином, завдадуть фінансові збитки. Також конкуренти можуть знайти в отриманої інформації факти порушення законодавства та передати їх в відповідні державні органи.

Одним словом, якщо в компанії постало питання про захист комунікацій - насамперед необхідно захищати пошту, так як основна маса документів пересилається через неї.

Безкоштовні поштові сервіси можуть також відстежувати ваші електронні листи і контакти, щоб показувати вам таргетовану рекламу.

Саме тому багато компаній використовують власні сервіси для обміну повідомленнями всередині компанії, адже базу даних локального сервера набагато важче зламувати, адже до неї нема прямого доступу.

Існують і з більш тривіальні причини використання корпоративної пошти, які, тим не менш, відіграють велику роль у формуванні образу компанії.

Крім того, що почта компанії використовується як зручний іінтерфейс для комунікації співробітників компанії із клієнтами, це також чудовий інструмент рекламної кампанії. Кожне повідомлення відправлене співробітниками, буде нагадувати клієнтам яку саме компанію вони представляють.

Отже, темою даної бакалаврської дипломної роботи було обрано розробка сервісу для відкладеної відправки поштових повідомлень необхідної для масової розсилки поштових повідомлень. Робота складається зі вступу, чотирьох розділів, та висновками по проекту. У першому розділі проведене дослідження вже існуючих рішень реалізації сервісу обміну повідомлень, у другому описані обрані технології для реалізації та спроектовані основні вимоги до системи. У третьому описано реалізація всіх складових сервісу, у четвертому розділі, розписано методи та результати тестування сервісу. У висновках проведено підсумки по проведеній роботі.

РОЗДІЛ 1.

ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ЇХ АНАЛІЗ

1.1. Основні поняття та структура сервісів обміну повідомлень

Сервіс обміну повідомлень – це великий портал, організований в виді програмного продукту, призначений для обміну інформації між користувачами, як сервісу що використовується , так і, можливо, інших таких сервісів.

Найрозповсюдженішою структурою сервісів обміну повідомлень є структура поштових сервісів. Вона складається з:

1. Серверу на якому зберігається БД зі всіма листами та інформацією про користувачів.
2. Програм клієнтів за допомогою яких можна отримати інформацію, що зберігається на сервері.

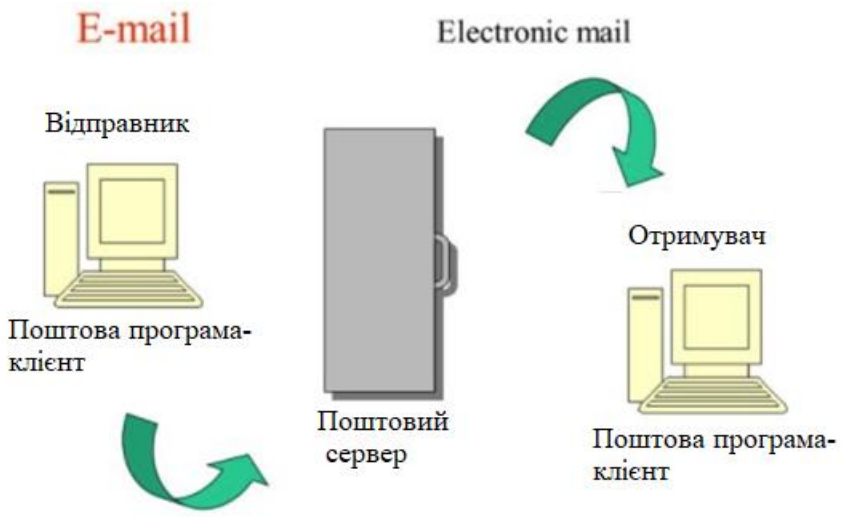


Рис. 1.1 Архітектура поштового сервісу [1]

В залежності від розташування серверу їх поділяються на:

1. Пошта на хостингу

Поштові сервера належать і адмініструються хостинг-оператором.

Поштова система обслуговує поштовий сервіс великої кількості доменів одночасно.

Всі користувачі пошти підключаються до серверів однаково.

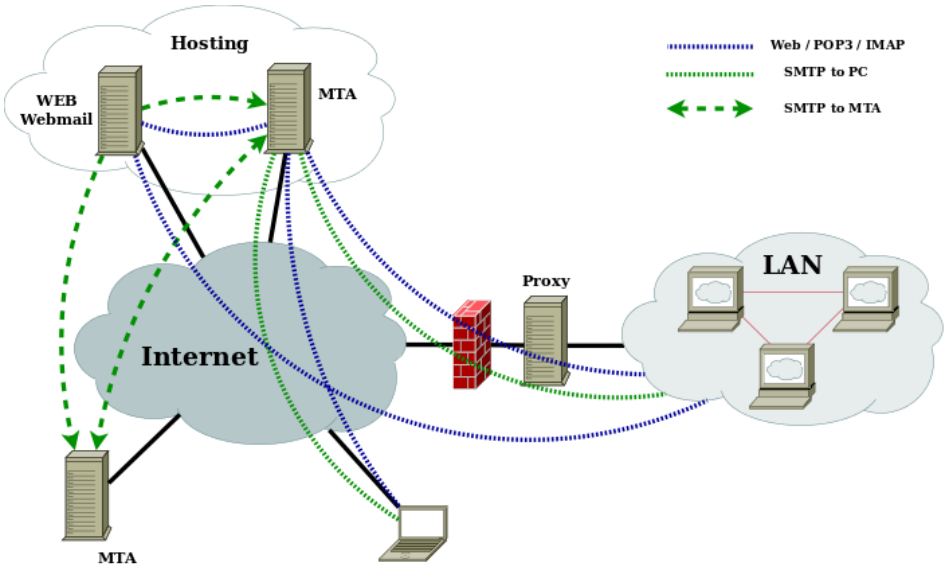


Рис. 1.2. Архітектура поштового сервісу на хостингу [2]

Переваги:

- Поштова система домену розташована на обладнанні хостинг-оператора.
- Відправлення та отримання E-Mail-кореспонденції, однакова для всіх користувачів домену і не залежить від територіального розташування користувача.
- Проблема спаму для користувачів пошти такого домену, як правило, відсутня, або її вирішують засобами і способами, пропонованими хостинг-оператором з прийнятною ефективністю.

Недоліки:

- Розмір поштової скриньки користувача обмежений хостинг-оператором, і як правило, невеликий. Це змушує деяких користувачів використовувати протокол POP3. В результаті архів пошти на хостингу не зберігається, що при WEB-доступі до поштової скриньки не дозволяє бачити всю історію листування.
- Використання своїх інструментів і сервісів для фільтрації спаму неможливо. Інструментарій управління поштою домену повністю залежить від хостингу.

2. Поштовий сервер (а) на території компанії. Зовнішніх користувачів пошти немає.

Всі користувачі поштової системи домену розташовані в локальній мережі компанії (LAN).

Зовнішні користувачі поштового сервісу, або підключаються до LAN компанії по VPN, або користуються WEB-доступом.

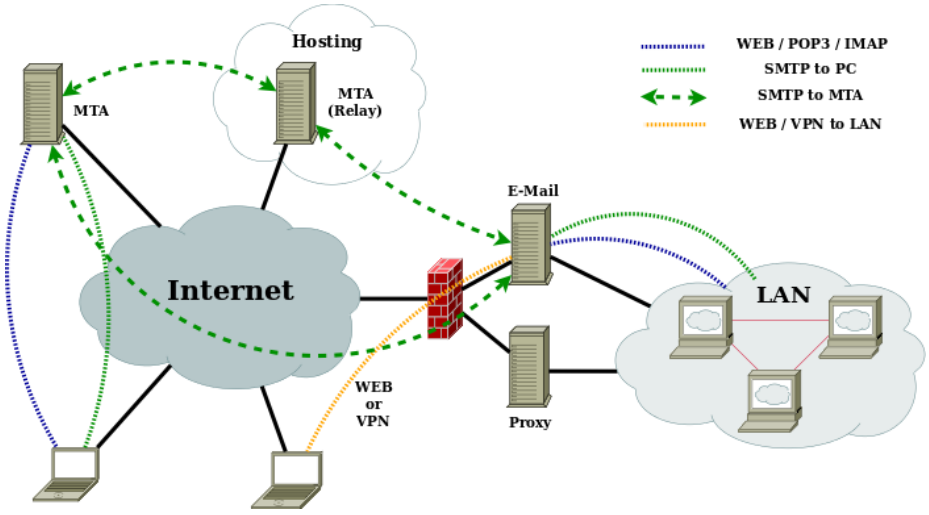


Рис. 1.3. Архітектура поштового сервісу на території компанії, у випадку коли немає зовнішніх користувачів пошти [2]

Переваги:

- Поштова система домену, розташована на території і під контролем компанії.
- Можливість гнучкого налаштування поштової системи. У тому числі різних антиспаму і антивірусного ПЗ.
- Поштовий сервер хостинг-оператора не використовується або використовується як допоміжний Relay для вхідної пошти, а так же, в деяких випадках, і для вихідної кореспонденції.
- Розмір поштової скриньки користувача визначається внутрішньою політикою компанії. Сервер має POP3, IMAP і WEB доступ до поштової скриньки.

Недоліки:

- Для супроводу E-Mail-сервера необхідний адміністратор(и) і технічні ресурси (сервер, живлення, надійні комунікації і т.д.).
- Користувачі E-Mail поза LAN і не підключені до офісної мережі по VPN можуть користуватися тільки WEB-поштою. Використання SMTP протоколу для відправки кореспонденції важко, тому що більшість користувачів підключається з динамічних адрес різних інтернет провайдерів.

3. Універсальна архітектура поштової системи

Територіальне розташування користувачів пошти не регламентовано, або визначається внутрішньою політикою і правилами компанії.

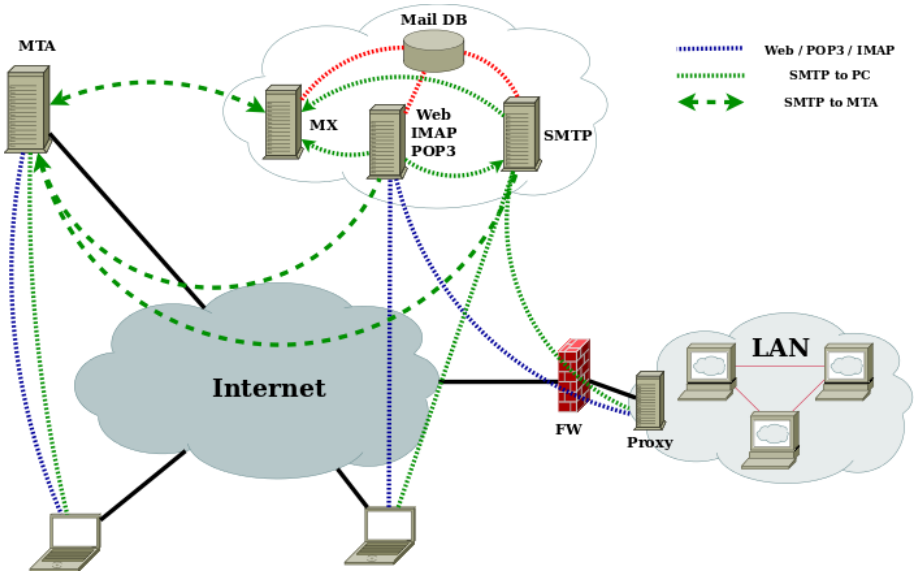


Рис. 1.4. Універсальна архітектура поштової системи [2]

Переваги:

- Поштова система забезпечує однаковий доступ усім користувачам поштової системи домену, як всередині локальної мережі компанії, так і поза нею.
- Користувачі, для доступу до пошти, можуть використовувати будь-яке доступне ПЗ.

Недоліки:

- Велика вартість обладнання та складність системи.

Важливо розуміти, що робота електронної пошти складається з трьох пов'язаних, але відокремлених етапів:

1. Прийом вхідної кореспонденції з поштових серверів інших доменів.

За прийом кореспонденції відповідає сервер вхідної пошти, тобто сервера зазначені в MX-записах DNS-зони домену (див. рис. 1.4 сервер MX).

Сервер вхідної кореспонденції не повинен приймати прямі підключення кінцевих користувачів пошти доменів, які він обслуговує.

2. Відправка користувачами вихідної кореспонденції.

Відправлення здійснюється з поштового клієнта або WEB-клієнта, по протоколу SMTP на сервер вихідної пошти MTA (MTA - Mail Transfer Agent), який в свою чергу здійснює її доставку на поштові сервери доменів призначення (див. рис. 1.4 Сервер SMTP), як окремий випадок, пересилає її на MX-сервер домену, подібна операція дозволяє спростити настройку домену, але підвищує навантаження на MX-сервер.

SMTP-сервер, якщо він не пересилає повідомлення на інший довірений Relay (наприклад: на сервер MX), а займається доставкою повідомлень самостійно, повинен бути зазначений в SPF-записи DNS-зони домену, тобто бути заявлений як довірений відправник електронної кореспонденції домену.

Сервер вихідної кореспонденції, приймає прямі підключення кінцевих користувачів пошти і виконує Relay їх відправлень. Отриманням і доставкою вхідної, для домену, кореспонденцією він не займається.

3. Доступ користувача до кореспонденції на сервері.

Отримання і перегляд кореспонденції здійснюється за протоколами WEB / POP3 / IMAP (див. рис. 1.4 Сервер Web / IMAP / POP3).

При використанні сервісів RBL / DNSBL поєднання серверів вхідної та вихідної кореспонденції, тобто серверів MX і SMTP (див. рис. 1.4) - неприпустимо. Всі інші варіації суміщення і рознесення серверів вхідної та вихідної кореспонденції, а також доступ до поштових скриньок, на розсуду адміністратора поштової системи.

Коли поштовий сервер один, але необхідно виконати формальний поділ MX і SMTP серверів, то для вирішення завдання використовується звичайний TCP / IP Redirector (redir, tcpredir, pen і т.п.). Основна мета, перенаправити підключення клієнта на поштовий сервер MX, з адреси, гарантоване не блокується сервісами RBL / DNSBL. Зазвичай редиректор встановлюється на сам поштовий сервер або сусідній сервер (наприклад: Proxy).

Важливо пам'ятати:

Для зовнішніх користувачів в якості сервера вихідної пошти вказується адреса і порт редиректора.

Так, якщо адрес у компанії небагато, то можна використовувати адресу основного поштового сервера, але не стандартний порт, наприклад: порт 2525.

Якщо використовується додатковий адресу, то порт краще залишити стандартний - 25-ий.

Підключення зовнішніх користувачів, при відправленні кореспонденції, проходить через редиректор, з адреси який на поштовому сервері виключений зі списку клієнтських адрес (тобто з нього не допускається відправка кореспонденції, на зовнішні або всі домени, без авторизації), але при цьому, він не влучає у блокуються сервісами RBL / DNSBL адресні діапазони.

Якщо редиректор встановлюється на поштовий сервер, то потрібно враховувати, що більшість MTA всі свої локальні адреси вважають довіреними, і не всі MTA дозволяють змінити статус цих адрес до рівня вимагає авторизації.

Іншою структурою сервісів для обміну повідомлень є структура месенджерів.

Instant Messaging (IM, миттєві повідомлення, Інтернет-пейджер) альтернативний до електронної пошти засіб спілкування, що за останні 10 років почав стрімко набувавти популярніть. Прообразом IM були онлайн-чати, які використовуються з початку 1990-х років (з них було взято основний принцип роботи - миттєва доставка повідомлень від співрозмовника до співрозмовни-

ка). Рішення мало клієнт-серверну архітектуру (стало класичним для подібних систем) - користувач завантажував безкоштовну програму-клієнт, яка підключалася до сервера, на якому зберігалися реєстраційні дані (присвоєний системою шестизначний номер і пароль) і список контактів.

Основна відмінність миттєвих повідомлень від інших засобів обміну повідомлень полягає у тому, що за рахунок постійного підтримання зв'язку між користувачами, обмін повідомленнями відбувається у реальному часі, тобто відправлене повідомлення у дуже короткий термін надходить до отримувачів.

Більшість мереж служби миттєвих повідомлень використовують закриті або пропрієтарні протоколи (власні протоколи, що належать тільки одній мережі) обміну інформацією. В основному в кожній з таких мереж застосовується свій месенджер.

Між різними мережами IMS зазвичай відсутні взаємозв'язку, тому месенджер однієї мережі, наприклад Telegram не може зв'язатися з месенджером мережі Skype. Це означає, що для ведення спілкувань між собою користувачі повинні зареєструватися в одному і тому ж сервісі і встановити їх месенджери.

Кожна система миттєвого обміну повідомленнями повинна мати такі складові:

- Система адресації (ідентифікації) клієнтів.
- Система обліку стану клієнтів, що перевіряє підключення користувачів.
- Система доставки повідомлень, що представлена різними протоколами передачі інформації (зазвичай месенджери мають свій власний протокол, тому обмін повідомленнями між клієнтами різних месенджерів не підтримується).

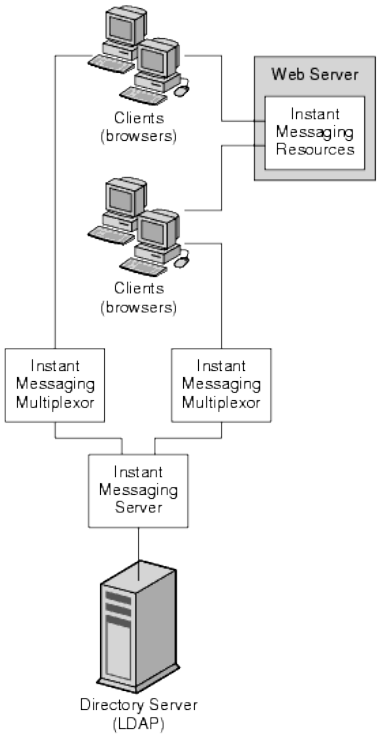


Рис 1.4. Базова архітектура сервісів миттєвих повідомлень [3]

Переваги:

- Обмін повідомленнями відбувається в реальному часі
- Можливість створювати чати, конференції.
- Підтримка відображення файлів різних форматі з клієнтської програми.

Недоліки:

- Велика вартість обладнання та складність системи.

Для кращого представлення про особливості, плюси та мінуси поштових сервісів зробимо їх порівняльну характеристику (Табл 1.1.).

Таблиця 1.1. Порівняння існуючих поштових сервісів

Назва	Gmail	Яндекс.Пошта	Yahoo!
Ціна	Безкоштовно для особистих потреб Платно для бізнесу, від 5 $ за користувача в місяць.	Безкоштовно для будь-якого користувача	Безкоштовно для будь-якого користувача
Робота з різних пристроїв	+	–	–
Повідомлення	+	+	–
Пошук	+	–	–
Обсяг сховища	15 ГБ	10 ГБ	1 024 ГБ
Експорт / імпорт контактів	–	+	+
Імпорт пошти з іншого ящика	+	+	–
Прив'язка свого домену	+	+	–
Перегляд документів	–	+	+
Делегування доступу	+	+	–
Перевірка правопису	–	+	–
Максимальний розмір вкладень	25 Мб	–	25 Мб
Автовідповідачі	+	+	–
Персональні дані	Збираються	–	–

Звичайно для більшості випадків цих сервісів цілком достатньо, однак для нашого випадку цілком очевидно, що краще мати повний контроль над системою і мати можливість її повного налаштування під власні потреби, адже неможливо довести, що жодний з цих сервісів не буде дискредитований працівниками компанії, яка володіє сервісом.

Сервіс обміну даних, якщо не з самого початку, то з часом, стає все більш складним, з'являються додаткові опції для керування самим як процесом обміну даних, так і процесом підготовки інформації, тим самим ускладнюється інтерфейс користувача, а сервіс розвиває внутрішню структуру. Одною з таких опцій є можливість автоматизованої відправки повідомлень.

1.2. Опис основних рішень автоматизованої розсилки листів

1.2.1. Область застосування

При організації масової розсилки, що складається з декількох повідомлень, часто буває важливо доставити листи адресатам зі строго певною періодичністю. Припустимо, кілька листів в рамках рекламної кампанії повинні бути відправлені з проміжком в 3 дня опівночі (наприклад, якщо отримувачі живуть в іншому часовому поясі). Щоб зробити розсилку вчасно, буде потрібна допомога щоденника і можливість відправити повідомлення саме в цей час.

Щоб не займатися відправкою листів опівночі і не забути про необхідність зробити розсилку, найпростіше скористатися можливістю відкладеної відправки. Складіть текст повідомлення, виберіть дату і час повідомлення, вкажіть одержувачів і займіться іншими справами - лист буде відправлено без вашої участі в точно зазначений момент часу.

1.2.2. Очікування дедлайну на сервері

Найбільш розповсюдженішою моделлю автоматизованої розсилки листів є розсилка завчасно збережених на сервері листів, або їх шаблонів, в заданий час (дедлайн) та по заданій адресі, які також збережено в базі даних.

Така модель отримала свою популярність через свою простоту в створенні та те що більшість випадків, коли необхідно відправляти повідомлення в строго заданий час, цілком можуть буди покриті цією моделлю, і що найбільш корисне в якості розширення вже існуючих інтерфейсів для відправки повідомлень не перенавантажуючи їх.

Типовий сценарій створення автоматизованої розсилки.

1. Користувач створює новий лист.
2. Користувач активує опцію відкладеної відправки, вносить дату та час відправки, та підтверджує відправку.
3. Система зберігає лист та супровідні дані (дата, час, адреса отримувача) на сервері.
4. До моменту відправки листа даний лист відображається в клієнтській програмі як чорновий варіант, який можна редагувати.
5. Система надсилає лист і знімає можливість його редагування для користувача.

Таким чином автоматизована розсилка мало чим відрізняється від звичайної розсилки, за виключенням того, що потрібно вибрати відповідну опцію в інтерфейсі програми та встановити час.

1.2.3. Очікування події зі сторони клієнтської програми

Досить часто повідомлення не можуть бути збереженні на сервері в момент створення листа з причин пов’язаних з неможливістю збереження незавершеної інформації. Наприклад потрібно відправити результат довготривалої обробки-аналізу даних в момент коли будуть сформовані дані для надсилання.

Для вирішення цієї проблеми існує спосіб відправлення листа використовуючи лише клієнтську програму без задіювання сервера на якому зберігаються проміжні дані.

Однак такий спосіб має багато недоліків:

1. Всі проміжні дані зберігаються на пристрої користувача,

2. Так як сама процедура надсилання листа і отримування листа проводиться клієнтськими програмами, в момент коли настає час відправки, клієнтські програми, а отже і пристрої, повинні бути активними.
3. Проблема маршрутизації для передачі даних також ставиться на користувача, який повинен, наприклад, створити локальну мережу з пристроями адресатів.

Саме тому таку архітектуру в чистому виді не використовують. Натомість використовують змішаний метод, де сервер використовується для відправки листа, а клієнтська програма очікує дедлайну (або іншої події) для відправки листа на сервер.

Типовий сценарій створення автоматизованої розсилки для ситуацій коли не можна відразу загрузити лист на сервер.

1. Користувач створює новий лист.
2. Користувач активує опцію відкладеної відправки.
3. Користувач вибирає подію за якою буде визначатись коли буде відправлено лист, наприклад момент коли буде створено файл з заданою назвою за заданим шляхом.
4. Клієнтська програма моніторить появу заданої події.
5. Клієнтська програма надсилає лист на сервер в момент коли відбувається вказана подія.
6. Система надсилає лист.

Таким чином не потрібно, щоб клієнтська програма в момент відправки була активна, а питання передачі листа залишається на стороні сервера.

Висновки до першого розділу

В даному розділі було проведено дослідження з огляду існуючих сервісів обміну повідомлень а саме: визначено поняття та структура сервісів обміну повідомлень, зокрема поштових сервісів та месенджерів, наведена порівняльна характеристика найпопулярніших поштових сервісів.

Також була сформульована область застосування автоматизованої розсилки повідомлень, та визначені існуючі методи їх реалізацій на серверній та клієнтській частині системи.

На основі отриманих даних можна діти до висновку, що не дивлячись на те що месенджери набирають все більшу популярність, розсилка повідомлень за допомогою поштових сервісів досі є найрозповсбдженішим засобом обміну повідомлень в бізнес середовищі. На основі цього можна припустити, що дана тема має актуальність, адже реалізована тема дипломної роботи надає можливісьть відсилати повідомлення в заданий момент часу, і при цьому не задіювати сторонні сервіси, таким чином зменшити вірогідність розсекречення інформації яка зберігалась в листі.

РОЗДІЛ 2.

ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Функціональні вимоги до системи

Для визначення технологій які будуть застосовані в проектуванні та розробці архітектури системи необхідно визначити та описати її функціональні вимоги.

- реєстрація та вхід у створений акаунт.
- коректна взаємодія клієнтських запитів на рівні заповнення форм.
- можливість переглянути створені листи.
- можливість переглянути отримані листи.
- можливість оновлення моделі на сервері.
- можливість редагування даних у БД.

2.2. Вибір технологій та бібліотек

2.2.1. Вибір мови програмування

Як можна зрозуміти з заданих функціональних вимог до системи сервіс повинен мати клієнт-серверну архітектуру, від цього і потрібно відштовхуватись при виборі існуючих технологій для майбутньої реалізації веб-додатку.

Мова програмування Python хоч і не створювалась спеціально під розробку веб-додатків, однак, має багато розвинених бібліотек та фреймворків пов’язаних з рішенням веб-завдань від об'єктно-реляційного відображення до веб-скрепінгу. Тобто Python виступає в ролі бази для об’єднання вже готових технологічних рішень, використовуючи які процес створення веб додатку перетворюється в нескладну та швидку задачу.

2.2.2. Вибір фреймворку

Під час написання веб-програми, окрім написання коду, що працює за "бізнес логікою", необхідно розібратися, наприклад, за якою URL-адресою

працює код, а також подбати про такі речі, як безпека, сеанси та розробка привабливих та функціональних сторінок HTML.

Можливо, для веб-сервісу нам потрібно надавати відповідь за допомогою JSON замість HTML-сторінки. Або нам потрібно і те і інше. Незалежно від того, чим займається наша програма, є частини, які цілком можливо можна використовувати в абсолютно інших програмах. Ось що таке веб фреймворк: набір функцій, спільних для широкого кола веб-додатків. [4]

Набір функцій які надає фреймворк може сильно відрізнятись від інших фреймворків. Деякі фреймворки пропонують безліч функціональних можливостей, включаючи маршрутизацію URL-адрес, системи шаблонів HTML, ORM для взаємодії з реляційними базами даних, безпеку, сеанси, генерацію форм тощо. Їх іноді називають фреймворками повного стека.

Інші фреймворки, відомі багатьма як мікро фреймворки, пропонують набагато менш різноманітний набір функцій і зосереджуються на простоті. Зазвичай вони пропонують маршрутизацію URL-адрес, шаблони та інший базовий функціонал.

Існує багато веб-фреймворків для Python. Крім розміру та рішень створених для розробника, багато з них пропонують унікальні функції або особливості функціоналу щодо того, як має робити веб-додаток. Деякі розробники одразу відчують потяг до якихось фреймворків або після деякого аналізу роблять висновок, що один із них краще підходить для конкретного проекту. Незалежно від обрангог фреймворку, завжди добре розуміти різноманітність інших доступних фреймворків, щоб при необхідності можна було зробити кращий вибір.

Django - це, без сумніву, найпопулярніший веб-фреймворк для Python на момент написання цього тексту. Django - це система високого рівня, створена для того, щоб піклуватися про найчастіші потреби веб-додатків. [5]

Сайт на Django зазвичай будується із одного або кількох частин, іменуємих додатками, один проект може підтримувати відразу декілька додатків, що і відрізняє цей фреймворк від деяких інших.

Архітектура Django подібна на «Модель-Вигляд-Контролер» (MVC), тому Django дуже добре підходить для веб-додатків, керованих базами даних. Він не тільки включає власне об’єктно-реляційне відображення (ORM), але може робити автоматичну генерацію форми на основі схем і навіть допомагає при міграціях. Після того, як ваші моделі визначені, для доступу до ваших даних можна використовувати багатий API Python. Django також пропонує динамічний адміністративний інтерфейс, який дозволяє аутентифікованим користувачам додавати, змінювати та видаляти об’єкти.

Адміністративний модуль дозволяє створювати, змінювати і вилучати будь-які об'єкти наповнення сайту, протоколюючі всі дії, а також надає інтерфейс для управління користувачами і групами (з призначенням прав). Це дає змогу мати на сторінку адміністратора на початку циклу розробки, а також розпочинати заповнення даних та тестування моделей, поки формуються користувацькі частини програми.

2.2.3. Вибір інструментів для стилізації шаблонів.

Для створення адаптивних шаблонів сторінок було вирішено використовувати Bootstrap. Bootstrap [6] - це відкритий і безкоштовний HTML, CSS і JS фреймворк, який використовується веб-розробниками для швидкої верстки адаптивних дизайнів сайтів та веб-додатків.

На Bootstrap створено дуже багато різних сайтів, і їх можна як приклад подивитися на них можна на сторінці Bootstrap Expo.

Основна область його застосування - це розробка фронтенда сайтів і інтерфейсів адмінок. Серед аналогічних систем (Foundation, UIkit, Semantic UI, InK і ін.) Фреймворк Bootstrap є найпопулярнішим.

Популярність Bootstrap пов'язана з тим, що він дозволяє верстати сайти в кілька разів швидше, ніж це можна виконати на «чистому» CSS і JavaScript. А в нашому світі час - це найдорожчий ресурс. Також його популярність пов'язана з доступністю. Вона полягає в тому, що на ньому навіть початківець

розробник може верстати досить якісні макети, які важко було б виконати без глибоких знань веб-технологій і достатньої практики.

Фреймворк Bootstrap являє собою набір CSS і JavaScript файлів. Щоб його використовувати ці файли необхідно просто підключити до сторінки. Після підключення вам стануть доступні інструменти даного фреймворка: колоночная система (сітка Bootstrap), класи і компоненти.

Переваги, які дає фреймворк Bootstrap при розробці на його основі frontend частини сайтів і інтерфейсів адмінок:

- висока швидкість створення якісної адаптивної верстки навіть початківцями веб-розробниками (досягається це завдяки використанню готових компонентів, створених професіоналами);
- кроссбраузерність і кросплатформенність
- наявність великої кількості готових добре продуманих компонентів, протестованих величезним співтовариством веб-розробників на різних пристроях;
- можливість налаштування під свій проект, досягається це за допомогою зміни SCSS змінних і використання Bootstrap міксинів (можна змінити кількість колонок, кольору, радіуса заокруглень кутів елементів, відступи між колонками і багато іншого);
- низький поріг входження; для роботи з фреймворком не обов'язково мати глибокі знання з HTML, CSS, JavaScript і jQuery (досить знати тільки основи перерахованих вище технологій);
- наявність добре продуманого дизайну компонентів і узгодженості (в Bootstrap всі компоненти виконані в єдиному стилі);
- наявність величезної спільноти, великої кількості статей, рецептів і відеоматеріалів; все це при бажанні допоможе не тільки добре розібратися в фреймворку, але і знайти відповіді практично на будь-які питання.

Фреймворк Bootstrap - це проект з відкритим вихідним кодом, доступним на Github. Він має ліцензію MIT. Це означає, що його можна використовувати безкоштовно як в особистих, так і в комерційних проектах.

Однак на Bootstrap, звичайно, верстають далеко не все. Його не використовують, наприклад, для:

- створення фронтенда проектів з унікальним дизайном;
- розробки проектів, в яких замовник готовий платити за проект на «чистому» CSS і JavaScript (в більшості випадках така розробка здійснюється в команді, в якій кожен її учасник виконує якийсь свій певний набір функцій);
- верстки особистих проектів, якщо у вас є достатня кількість часу і ваш рівень знання технологій HTML, CSS і JavaScript достатній, щоб це здійснити.

Bootstrap, так і більшість подібних фреймворків, має недоліки. Серед них:

- більш великий розмір кінцевих CSS і JavaScript файлів, ніж якщо їх написати конкретно під цей проект на чистому CSS і JavaScript (це пов'язано з тим стилі і код JavaScript містять багато всього і на більшості сайтах потрібна тільки та чи інша частина);
- використання Bootstrap для верстки сайтів, в яких знадобиться чимало переписування його CSS і JavaScript коду (простими налаштуваннями Bootstrap змінних тут не обійтися).

2.2.4. Вибір технології взаємодії з БД

Django підтримує багато різних СУБД[7]. Зокрема до них відносяться PostgreSQL, MySQL, Oracle і SQLite.

Якщо ви зайняті розробкою простого проекту або чогось, що ви не плануєте розгорнути у виробничому середовищі, SQLite в цілому є найпростішим варіантом, оскільки він взагалі не вимагає установки окремого вебсервера. Однак, SQLite дуже відрізняється від інших баз даних, тому, якщо

ви працюєте над чимось суттєвим, рекомендується вибрати ту ж базу даних, яка буде використана на "бойовому" сервері.

SQLite працює в пам'яті та створює резервні копії даних у файлах на диску. [8] Хоча ця стратегія добре працює для розробника, якщо хостинг не має ефемерну файлову систему. В іншому випадку ви можете записувати в БД, і ви можете читати з неї, але вміст буде періодично очищатися. Якби ви використовували SQLite на Heroku, ви втрачали б всю свою базу даних хоча б раз на 24 години. Навіть якщо б диск був персистентнй SQLite все одно був би поганим вибором.

Heroku використовує dyno [9] – ізольовані, віртуалізовані контейнери Linux, призначеними для виконання коду на основі визначеної користувачем команди. Оскільки SQLite не працює як сервіс, кожне dyno буде виконувати окрему запущену копію. Кожна з цих копій потребує власного простору на диску. Це означатиме, що кожен dyno підтримуючи ваш додаток матиме різний набір даних, оскільки диски не синхронізовані.

СУБД PostgreSQL не має цього суттєвого недоліку тому вона цілком підходить для розробки системи.

Однак в цілому вибір СУБД мало позначається на коді програми, через те, що джанго підтримує вбудований Django ORM (Об'єктно-реляційне відображення) - бібліотеку кодів, яка автоматизує передачу даних, що зберігаються в реляційних таблицях баз даних, в об'єкти, які більш частіше використовуються в коді програми.

ORM надають абстракцію високого рівня на реляційну базу даних, яка дозволяє розробнику писати код Python замість SQL для створення, зчитування, оновлення та видалення даних і схем у своїй базі даних. Розробники можуть використовувати мову програмування, яка зручна для них, для роботи з базою даних замість написання SQL-операторів або збережених процедур.

Можливість запису коду Python замість SQL може пришвидшити розробку веб-додатків, особливо на початку проекту. Потенційне збільшення

швидкості розвитку відбувається через те, що не потрібно переходити з коду Python в написання декларативної парадигми SQL-операторів. Хоча деякі розробники програмного забезпечення можуть не проти перемикання між мовами вперед і назад, зазвичай простіше збити прототип або запустити вебдодаток за допомогою однієї мови програмування.

ORM також теоретично дозволяє перемикати додаток між різними реляційними базами даних. Наприклад, розробник може використовувати SQLite для локального розвитку та MySQL у виробництві. Виробнича програма може бути переключена з MySQL на PostgreSQL з мінімальними модифікаціями коду.

Однак на практиці найкраще використовувати ту саму базу даних для місцевого розвитку, яку використовують у виробництві. Інакше у виробництві можуть виникнути несподівані помилки, яких не було помічено в умовах місцевого розвитку. Також рідко буває, що проект перейшов би з однієї бази даних у виробництві в іншу, якщо не було нагальної причини.

Так наприклад, описаний нижче код класу User, після створення міграцій та синхронізації БД з проектом, автоматично буде створена таблиця з назвою яка выдповідаэ назві класу, і полями з типами приведеними до відповідних типів у даній БД.

```
    class User(models.Model):
        login = models.CharField(related_name="login",
max_length=100)
        password = models.CharField(related_name ="password",,
max_length=100)
        email = models.EmailField()
```

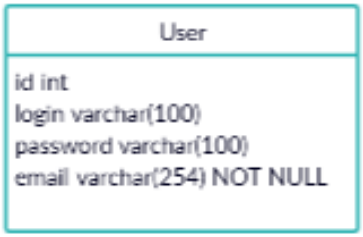


Рис. 2.1. Утворена таблиця за вказаною моделлю

Варто додати, що поле id таблиці було створено автоматично, так як жодне поле моделі не було об’явлене як primary key, а поле email таблиці за замовчуванням прийняла назву поля моделі.

2.3. Загальні положення побудови архітектури додатку

Django заохочує вільне зв'язування і суворий поділ частин програми. Якщо слідувати цій філософії, то легко вносити зміни в одну конкретну частину додатку без шкоди для інших частин. У функціях представлень, наприклад, надзвичайно важливо відокремлювати бізнес-логіку від логіки відображення за допомогою шаблонної системи. Використовуючи шар для роботи з базою даних, ми застосовуємо цю ж філософію для логіки доступу до даних.

Ці три речі разом - логіка доступу до даних, бізнес-логіка і логіка відображення - складають концепцію, яку називають шаблоном Модель-Представлення-Управління (Model-View-Controller, MVC)[10] архітектури програмного забезпечення. У цій концепції термін «Модель" визначає логіку доступу до даних; термін «Представлення» відноситься до тієї частини системи, яка визначає, що показати і як; а термін «Управління» відноситься до тієї частини системи, яка визначає яке представлення треба використовувати, в залежності від вводу користувача, по необхідності отримуючи доступ до моделі.

Метою чіткого визначення скорочень, подібних MVC, є упорядкування взаємодії між розробниками.

Django слідує моделі MVC досить близько, тобто, може бути названий MVC сумісним середовищем розробки. Ось приблизно як M, V і C використовуються в Django:

- M – доступ до даних, обробляється шаром роботи з базою даних.
- V –частина, яка визначає які дані отримувати і як їх відображати, обробляється представленнями і шаблонами.
- C – частина, яка вибирає представлення залежно від вводу для користувача, обробляється самим середовищем розробки, слідуючи створе-

ній схемі URL, і викликає відповідну функцію Python для зазначеного URL.

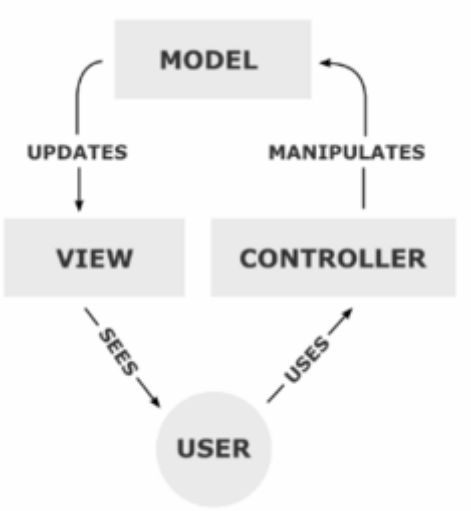


Рис. 2.2. Архітектура MVC[11]

Так як «С» обробляється середовищем розробки і все Django відбувається в моделях, шаблонах і представленнях, на Django посилаються як на MTV-орієнтоване середовище розробки. В MTV-підході до розробки:

- M визначено для «Моделі» (Model), шару доступу до даних. Цей шар знає все про дані: як отримати до них доступ, як перевірити їх, як з ними працювати і як дані пов'язані між собою.
- T визначено для «Шаблона» (Template), шару представлення даних. Цей шар приймає рішення щодо подання даних: як і що має відображатися на сторінці або в іншому типі документа.
- V визначено для «Представлення» (View), шару бізнес-логіки. Цей шар містить логіку, як отримувати доступ до моделей і застосовувати відповідний шаблон. Можна розглядати його як міст між моделями і шаблонами.

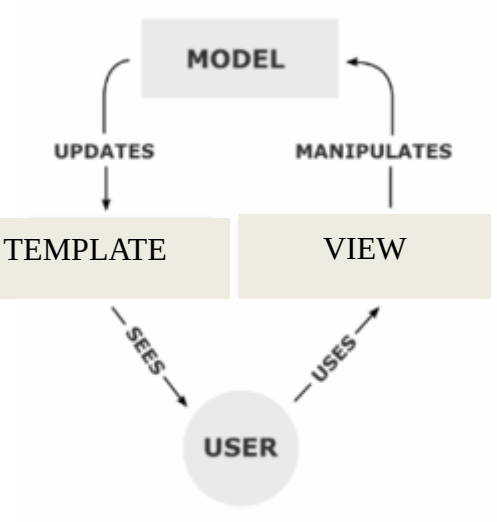


Рис. 2.3. Архітектура MTV

Таким чином представлення в Django потрібно розглядати як «контролери», а шаблони Django - як «представлення». Це сумна плутанина виникла в результаті різних тлумачень MVC. В інтерпретації Django «представлення» описує дані, які будуть представлені користувачеві. Неважливо як ці дані будуть виглядати, важливо які дані. Навпаки, в Ruby on Rails і подібних йому середовищах передбачається, що в роботу контролера включено прийняття рішення, які дані будуть представлені користувачеві, в той час як представлення точно визначає як ці дані будуть виглядати.

Жодна з цих інтерпретацій не має переваг над іншою. Важливо розуміти основну концепцію.

2.3.1. Веб сторінки

Для того щоб розпочати розробку веб-сторінок спочатку потрібно на основі функціональних вимог до системи визначити, які веб-сторінки, або їх частии, буде включати додаток.

- реєстрація – потребує окремої сторінки, або спливаюче вікно.
- вхід в акаунт – потребує окремої сторінки, або спливаюче вікно.
- коректна взаємодія клієнтських запитів на рівні заповнення наперед заданих форм – сторінка де користувач зможе створювати лист.
- можливість переглянути створені листи – сторінка де користувач зможе переглянути створені листи.

- можливість переглянути отримані листи – сторінка де користувач зможе переглянути отримані листи.
- можливість редагування даних у БД – сторінка адміністратора

Для представлення цих сторінок крайньо необхідним є використання мови шаблонів.

Система шаблонів Django надає теги, які функціонують аналогічно деяким програмам програмування - тег if для булевих тестів, тег для циклічного циклу тощо - але вони не просто виконуються як відповідний код Python, але і система шаблонів не виконуватиме довільні вирази мови Python. За замовчуванням підтримуються лише обмежена кількість тегів, фільтрів та синтаксисів (хоча можна додавати власні розширення до мови шаблонів за необхідності).

Не зважаючи на зручність мови шаблонів, не потрібно нею зловживати адже завжди потрібно пам’ятати, що система шаблонів Django не просто вбудований інтерпретатор Python в HTML. За задумкою: система шаблонів призначена для вираження презентації, а не програмної логіки, і використання її не в правильних цілях може значно сповільнити роботу програми.

Мова шаблонів має чотири основні скадові:

1. Шаблон - це текстовий файл. Він може генерувати будь-який текстовий формат (HTML, XML, CSV тощо). Шаблон містить змінні, які замінюються значеннями, коли шаблон викликається, та теги, які керують логікою шаблону
2. Змінні – Змінні виглядають так: {{ variable }}. Коли двигун шаблону стикається зі змінною, він обчислює цю змінну і замінює її результатом. Імена змінних складаються з будь-якої комбінації буквено-цифрових символів та підкреслення ("_"), але можуть не починатися з підкреслення. Крапка (".") Також з'являється у змінних розділах, хоча це має особливе значення, як зазначено нижче. Важливо, що ви не можете мати пробіли чи знаки пунктуації у назвах змінних. Для доступу до атрибутів змінної використовується символ крапки (.).

3. Фільтри – спеціальні модифікатори для зміни відображення змінних. Вихід одного фільтра може бути застосований як вхід для наступного. {{text | escape | linebreak}}
4. Теги виглядають приблизно так: {% tag%}. Теги складніші за змінні: деякі створюють текст у вивід, деякі керують потоком, виконуючи цикли або логіку, а деякі завантажують зовнішню інформацію в шаблон, який буде використаний пізнішими змінними. Для деяких тегів потрібні початкові та кінцеві теги (тобто {% tag%} ... вміст тегів ... {% endtag%}). Django поставляється з близько двох десятків вбудованих тегів шаблонів. Django пропонує близько шістдесяти вбудованих шаблонних фільтрів.

2.3.2. Збереження паролів на сервері

Для поштового сервісу безпека данних коритувачів є одним з нагальних питань з високим пріорітетом, так як взломавши акаунт може бути вкрадена таємниця листування.

Навіть великі компанії час від часу допускають втрату великої кількості персональних даних користувачів, коли зламувач викладає у мережу дані про паролі та логіни користувачів.

Розглянемо найпоширеніші методи зберігання паролів.

1. Зберігання в звичайному рядковому представленні.

Пароль користувача для автентифікації зберігається в базі даних. При вході в акаунт користувач вводить пароль після чого система перевіряє пароль на збіг з даними збереженими в БД.

Переваги:

– У випадку коли користувач забув дані для входу їх можна надіслати на пошту
– Простота в створенні форм для реєстрації

Недоліки:

– Низька безпека, якщо зламувач отримав копію БД він отримав доступ до кожного акаунту користувача.
– У випадку коли в БД зберігається адреса електронної з'являється велика вірогідність зламу електронної пошти користувача, так як багато людей використовує один пароль для входу в акаунт різних веб-сайтів.

2. Зберігання в зашифрованому вигляді.

Перед збереженням паролю для входу в БД він шифруються спеціальним алгоритмом шифрування.

Для прикладу наведемо псевдокод шифрування алгоритмом RSA

1. вибрають два великих простих числа p і q;
2. обчислюють: n = p * q, m = (p - 1) * (q - 1);
3. вибрають випадкове число d, взаємно просте з m;
4. визначають таке число e, для якого є істинним вираз:(e*d) mod (m) = 1;
5. числа e і n - це відкритий ключ, а числа d і n - це закритий ключ;

На практиці це означає наступне: відкритим ключем зашифровують повідомлення, а закритим - розшифровують. Пара чисел закритого ключа тримається в секреті.

6. розбивається шифруємий текст на блоки, кожен з яких може бути представлений у вигляді числа M (i);

Зазвичай блок беруть рівним одному символу і представляють цей символ в виді числа - його номера в алфавіті або коду в таблиці символів (наприклад ASCII або Unicode).

7. шифрування алгоритмом RSA здійснюється за формулою: C (i) = (M (i) e) mod n;
8. розшифровка повідомлення проводиться за допомогою формули: M (i) = (C (i) d) mod n.

Даний підхід заснований на володінні секретом. Секретом є алгоритм шифрування або ключ, який використовується разом з сучасним алгоритмом шифрування.

Шифрування паролів - оборотна операція. Секрет використовується для спотворення пароля, і тейсамий секрет може бути використана для відновлення оригінального пароля. Коли користувач задає пароль, збережений пароль розшифровується за допомогою секрету, і паролі порівнюються. Альтернативний підхід - зашифрувати наданий пароль за допомогою секрету і порівняти дві спотворені версії - збіг показує, що наданий правильний пароль.

Якщо користувачеві треба відновити пароль, збережений пароль розшифровується і доставляється користувачеві (зазвичай по електронній пошті).

Переваги:

– Дещо більша степінь безпеки від зламу в порівнянні з простим збереженням паролів в БД.
– Для повного зламу потрібно мати відкриваючий ключ алгоритму шифрування

Недоліки:

– Витрачається час на шифрування.
– Якщо зламувач отримав копію даних БД та знає відкриваючий ключ алгоритму шифрування він отримав доступ до всіх акаунтів користувачів.
– У випадку коли багато користувачів мають ідентичні паролі, їх зашифровані представлення також будуть ідентичні. Таким чином навіть не знаючи відкриваючого ключа з отриманої таблиці паролів користувачів можна виділити найчастіше повторювані і розшифрувати їх методом підбору уживаних паролів, наприклад, 123456 qwerty, password1, або маючи дані з таблиці з підказками до паролю на їх основі зробити припущення про часто уживаний пароль.

3. Зберігання хеш-коду паролю.

Метод збереження паролю схожий на метод з використування шифрування, однак замість алгоритму шифрування використовується алгоритм хешування. Різниця в тому, що хешування не оборотна операція, тому отримати пароль після спотворення не є можливим, принаймні 100 відсотковою

ймовірністю. Тому тут використовують лише альтернативний підхід описаний вище.

Псевдокод алгоритму хешування sha-1:

Для повідомлення довільної довжини l, що не перевищує 2 ^ 64 біт, алгоритм SHA-1 формує 160-бітний хеш-образ. Процедура формування хеш-образу складається з наступних кроків.

1. Весь вихідний текст розбивається на блоки по 512 біт. У разі якщо довжина вихідного тексту не кратна 512 бітам, проводиться його вирівнювання за рахунок додавання в кінець біта зі значенням 1, m нульових бітів і 64-бітного представлення значення довжини вихідного повідомлення.
2. Ініціалізується п'ять 32-бітових робочих змінних

   A = 0x67452301; B = 0xefcdab89; C = 0x98badcfe;

   D = 0x10325476; E = 0xc3d2e1f0;

3. Виконується обробка чергових 512 біт вихідного тексту. Для цього значення змінних A, B, C, D, E копіюються в змінні a, b, c, d, e і далі для t від 1 до 80 виконується перетворення значень даних змінних за наступною схемою:

   TMP = (a<<<5) + $f_t(b,c,d) + e + W_t + K_t$;

   e = d;

   d = c;

   c = b<<<30;

   b = a;

   a = TMP;

,де нелінійна функція $f_t(b,c,d)$ та параметр K_t залежать від циклу ітерації, і приймають вигляд

$$f_t(b,c,d) = \begin{cases} (b\&c)|(!\,b\&d) & t = \overline{1,20} \\ b \oplus c \oplus d & t = \overline{21,40} \\ (b\&c)|(b\&d)|(c\&d) & t = \overline{41,60} \\ b \oplus c \oplus d & t = \overline{61,80} \end{cases} \quad K_t = \begin{cases} 0x5A827999 & t = \overline{1,20} \\ 0x6ED9EBA1 & t = \overline{21,40} \\ 0x8F1BBCDC & t = \overline{41,60} \\ 0xCA62C1D6 & t = \overline{61,80} \end{cases}$$

а «&» - побітова операція «І», « | » - побітова операція «АБО», «!» - операція побітового інвертування, «⊕» - операція побітового додавання за модулем 2, «<<<»– операція циклічного зсуву, «+» – операція додавання по модулю 2^{32}, W_t - одне з шістнадцяти 32-бітних слів 512-бітного блоку повідомлення при яких значення, яке визначається відповідно до наступним виразом:

$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1 \;\; t = \overline{17,80}$

4. Значення змінних a, b, c, d, e незалежно один від одного складаються по модулю 2^{32} зі значеннями змінних A, B, C, D, E, в які потім і поміщаються отримані результати.
5. Кроки 3-4 виконуються доти, поки не буде оброблений весь текст.

   Після обробки останнього блоку тексту значення хеш-образу формується як ABCDE.

Переваги:

– Розумний компроміс між часом пошуку таблицею і займаною пам'яттю

Недоліки:

– Витрачається час на хешування.
– Як і випадку з методом збереження в зашифрованому вигляді ідентичні паролі мають ідентичні хеш-коди, що спрощує метод підбору.
– Компанія Google проводить індексацію алгоритмів хешування, тому використання вже готових алгоритмів в чистому виді є недопустимим рішенням, так як доволі легко отримати пароль маючи хеш підібравши алгоритм зі збережених компанією Google, або у крайнє простому випадку просто «загуглити» хеш. Також в мережі інтернет доволі просто знайти райдужні таблиці, які використовуються для розкриття паролів, перетворених за допомогою важкозворотної хеш-функції, що пришвидшить злам всіх паролів збереженизв БД.

4. Зберігання хеш-коду паролю з застосуванням солі.

Метод майже ідентичний до методу зберігання хеш-коду паролю, за деякими відмінностями. Після застосуванням функції хешування на паролі

користувача, отриману послідовність видозмінюють за допомогою приєднування рядка випадкових символів (сіль). Отриману хеш-послідовність зберігають в БД разом з сіллю

Для автентифікації користувача потрібно повторити цю процедуру використовуючи сіль збережену в БД і перевірити отриману послідовність з наявною в БД.

Переваги:

– Розумний компроміс між часом пошуку таблицею і займаною пам'яттю
– Ідентичні паролі мають різну хеш-послідовність.
– Стійкість до зламу з застосуванням райдужних таблиць.[12]

Недоліки:

– Більший час обчислення хеш-послідовності.

Для заданого проекту через великі вимоги до безпеки буде використовуватись останній варіант збереження паролів, а саме зберігання хеш-коду паролю з застосуванням солі.

Висновки до другого розділу

У цьому розділі було виконано огляд функціональних вимог до сервісу відкладеної відправки поштових повідомлень. Також було наведено перелік використовуваних бібліотек та технологій для реалізації системи, і обґрунтовано причини переваги їх використання при реалізації проекту.

Було досліджено способи збереження паролів користувачів, і описані позитивні і негативні сторони кожного із основних способів захисту персональних паролів,

Таким чином на основі прийнятих рішень було визначена майбутня структура проету, а саме MTV – архітектура додатку, зробленого з допомогою фреймворку Django, де основні функції шару бізнес логіки будуть виконуватись з допомогою мови Python, а модель бази даних та всі SQL запити будуть реалізовані з допомогою функціоналу об’єктно-реляційного відображення, вбудованго у фреймворк.

На останок варто додати, що адаптивні шаблони веб-сторінок будуть створені з допомогою вбудованих, а також власне створених тегів бібліотеки jinja2, що є за замовчуванням вбудована у фреймворк, а їх стилізація буде проведена на основі стилів наданих бібліотекою Bootstrep, яка надає можливість стилізувати шаблони додатку не витрачаючи на це багато часу.

РОЗДІЛ 3.

РОЗРОБКА СИСТЕМИ

3.1. Огляд проекту

Для роботи проекту необхідно встановити дані бібліотеки, інструменти.

- Python 3.6.3 (мова програмування)
- Django 3.0.6 (веб-фреймворк)
- APScheduler 3.6.3 (бібліотека що дозволяє планувати виконання задань)
- django-multi-email-field 0.6.1 (бібліотека що дозволяє створити поле та віджет для зберігання списку електронних адрес у проекті Django)
- python-decouple 3.3 (бібліотека, що допомагає впорядкувати налаштування, щоб ви могли змінювати параметри, не потребуючи повторного перезапуску програми, а також сховати параметри налаштувань в інший файл)
- pydotplus 2.0.2 (бібліотека для автоматичного створення UML діаграм)
- pytz 2020.1 (бібліотека з допоміжними функціями для визначення часових поясів)
- django-registration 3.1 (додаток, який дозволяє зробити нестандартну реєстрацію користувача)
- JetBrains PyCharm Professional 2020.1.1 (інтегроване середовище розробки для мови Python)
- PostgreSQL 11.4 (Система управління базами даних)
- psycopg2 2.8.5 (бібліотека для підключення бази даних PostgreSQL до проекту)

Реалізація системи виконується на основі описаної в попередньому розділі тришарової MTV архітектури.

У процесі реалізації відбувається за рахунок розширення вже існуючої базової архітектури додатку на Django.

Архітектура проекту в середовищі розробки Pycharm наведена на наступному малюнку (Рис 3.1).

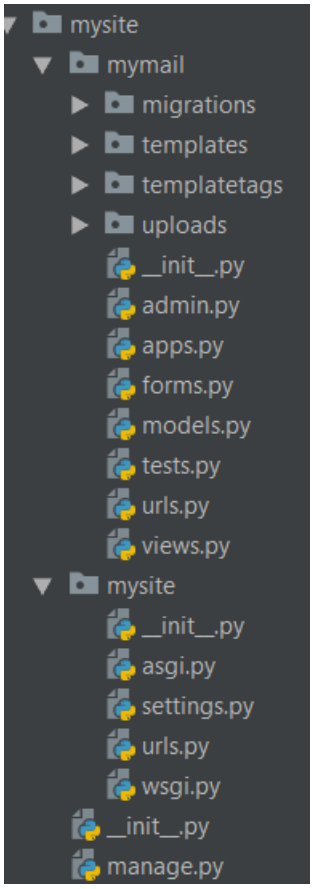


Рис. 3.1. Архітектура проекту в середовищі розробки

manage.py: Скрипт, який дозволяє вам взаємодіяти з проектом Django.

Внутрішній каталог mysite / - це пакет Python вашого проекту. Його назва - це назва пакета Python, яке ви будете використовувати для імпорту чогось з проекту (наприклад, mysite.urls).

- __init__.py і mysite/__init__.py: Порожній файл, який вказує Python, що поточний каталог є пакетом Python.
- mysite/settings.py: Налаштування/конфігурація проекту. Файл налаштувань Django містить повну конфігурацію встановленого проекту, такі як: вибір бази даних, локалізація, часовий пояс, пере направлення на url після реєстрації за замовчуванням.
- mysite/urls.py: Конфігурація URL-ів для вашого проекту Django. Це «зміст» усіх Django-сайтів.

- mysite/asgi.py: Точка входу вашого проекту для ASGI-сумісних веб-серверів.
- mysite/wsgi.py: Точка входу вашого проекту для WSGI-сумісних веб-серверів.
- В проекті визначений додаток mymail – це власне під проект в якому потрібно описувати поведінку програми.
- mysite/mymail/admin.py: Налаштування моделей які можна редагувати через інтерфейс адміністратора.
- mysite/mymail/apps.py: Застосування налаштувань проекту до додатку, визначення імені додатку.
- mysite/mymail/forms.py: Визначення користувацьких форм, які будуть застосовані для користувацького вводу.
- mysite/mymail/forms.py: Визначення вигляду моделі бази данних.
- mysite/mymail/tests.py: Тестування додатку.
- mysite/mymail/urls.py: Задання відповідності URL-ів до їх представлення (поведінки) для даного додатку..
- mysite/mymail/views.py: Визначення представлень для даного додатку..
- mysite/mymail/migrations: Каталог в якому зберігаються усі застосовані зміни до бази даних.
- mysite/mymail/templates: Каталог в якому зберігаються шаблони сторінок, тобто як вони будуть відображені в браузері.
- mysite/mymail/ templatetags: Каталог в якому зберігаються зареєстровані теги, які будуть використовуватись в додатку.

3.1.1. Опис url посилань та функцій їх представлень

Всього для проекту було створено 12 URL шаблонів, які можна побачити нижче.

- '' – не має шаблону, перенаправляє за посиланням 'received/0
- 'received/<int:page>' – відображається шаблоном search.html, за цим посиланням відображається список отриманих листів,

де <int:page> – число яке вказує, що відображаються останні повідомлення з номерами від 30*<int:page> до 30*(1+<int:page>)

- 'send/<int:page>' – ідентичний received/<int:page> за виключенням що відображаються відправлені повідомлення.
- 'templates/<int:page>' – ідентичний received/<int:page> за виключенням що відображаються повідомлення, які можна застосувати як шаблон для створення нових повідомлень.
- '<uuid:message_url>' – відображається шаблоном read_message.html та edit.html, сторінка на якій відображається повідомлення, якщо повідомлення вже відправлено то застосовується шаблон read_message.html, якщо ні, то якщо користувач є відправником edit.html, <uuid:message_url> – uuid4-код[13], унікальна послідовність з 32 шістнадцяткових символів, має значення ідентичне полю url екземпляру моделі Message, з його допомогою знаходиться потрібне повідомлення.
- 'new' – відображається шаблоном create_message.html, сторінка на якій можна створити повідомлення без початкових значень.
- 'new/<uuid:message_url>' – відображається шаблоном create_message. html, сторінка на якій можна створити повідомлення з початковими значеннями повідомлення з ідентичним до <uuid:message_url> полем url.
- 'delete/<uuid:message_url>' – не має відображення, повідомлення з полем url ідентичному <uuid:message_url> змінює поле show на значення False, якщо автентифікований користувач є відправником повідомлення. Якщо ні то змінює поле show екземпляру моделі Receivers, яке відповідає даному повідомленню та даному юзеру.
- 'just_delete/<uuid:message_url>' – не має віображення видаляє повідомлення з з полем url ідентичному <uuid:message_url> якщо поточний

користувач є відправником і поточний час менший за час поля send_date повідомлення.

- 'delete/<uuid:message_url>/Template' – не має відображення, повідомлення з полем url ідентичному <uuid:message_url> змінює поле show_template на значення False, якщо авторизований користувач є відправником повідомлення. Якщо ні то змінює поле show екземпляру моделі Receivers, яке відповідає даному повідомленню та даному юзеру.
- r'registration/' – відображається шаблоном registration.html, сторінка реєстрації нового користувача.
- r'logout/' – не має відображення, виходить з користувача.
- r'login/' – відображається шаблоном login.html, сторінка автентифікації користувача.

3.1.2. Опис інтерфейсу

Інтерфейс складається з основної панелі де знаходяться кнопки переходу на стронки сервісу, і контентного блоку нижче від панелі, де в залежності від сторінки знаходяться відповідні.

Основна панель має такий вигляд

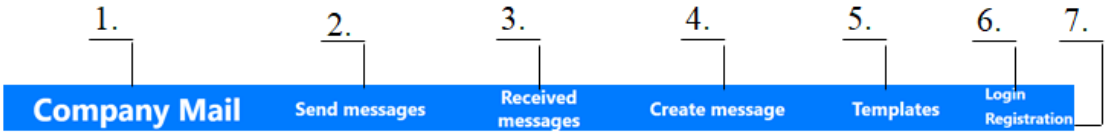


Рис. 3.2 Елементи основної панелі

Складаєься з таких кнопок:

1. Перехід за посиланням received/0/
2. Перехід за посиланням send/0/
3. Перехід за посиланням received/0/
4. Перехід за посиланням new/
5. Перехід за посиланням templates/0/

6. Перехід за посиланням login/, якщо користувач авторизований замінюється на текст з логіном користувача.
7. Перехід за посиланням registration/, якщо користувач авторизований замінюється на перехід за посиланням logout/

   При переході на сторінку send/<int:page>, або received/<int:page>, або templates/<int: page>, контентний блок такий вигляд

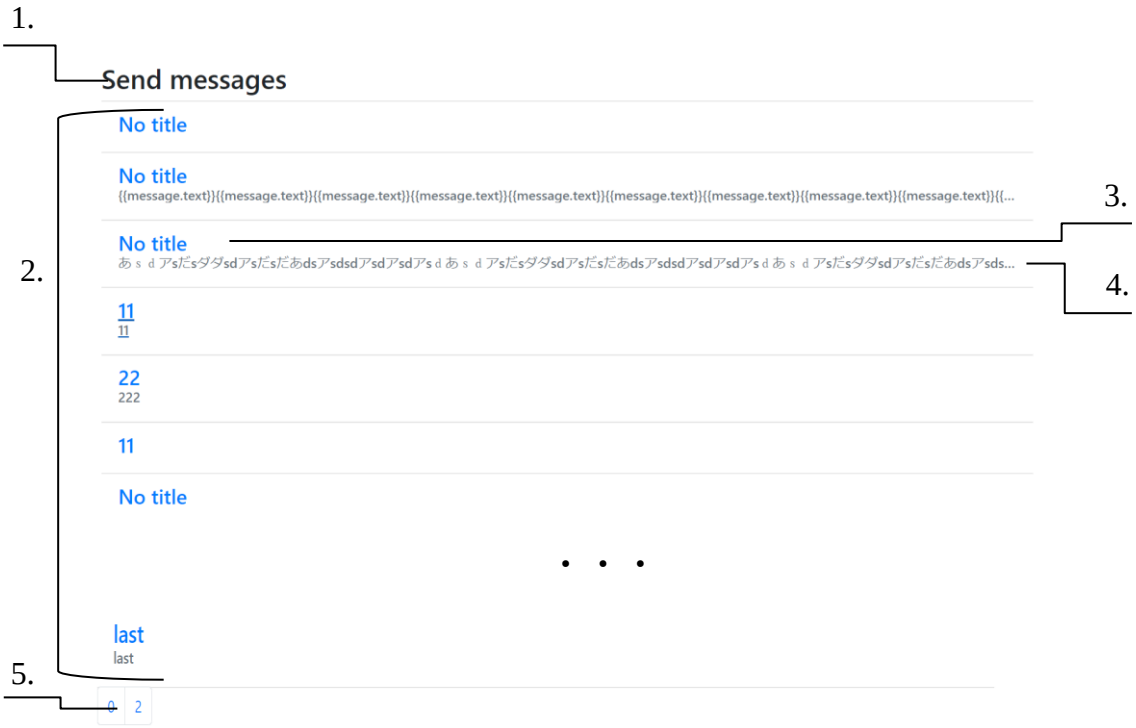


Рис. 3.3 Елементи контентного блоку сторінки перегляду списку отриманих повідомлень

1. Текст який відповідає на якій сторінці ви знаходитесь.
2. Список листів
3. Тема повідомлення (якщо користувач є отримувачем і лист прочитаний буде відображатись сірим кольором), (якщо користувач є відправником і лист вже відправлений буде відображатись сірим кольором),
4. Превью тексту повідомлення
5. Кнопки переходу на попередню та наступну сторінки.
6. При переході на сторінку <uuid:message_url>

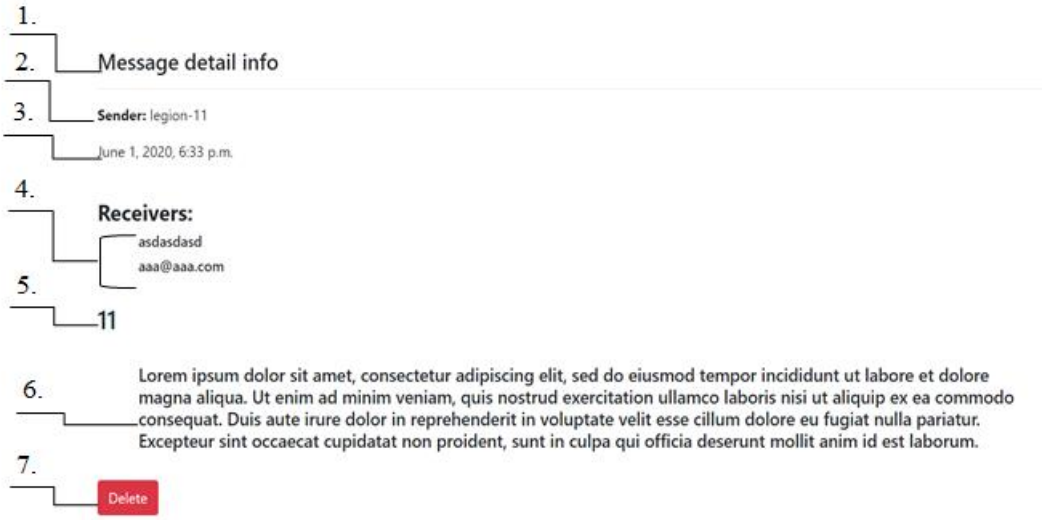


Рис. 3.4 Елементи контентного блоку сторінки перегляду повідомлення

1. Текст який відповідає на якій сторінці ви знаходитесь.
2. Логін користувача який відправив повідомлення.
3. Дата відправлення
4. Список отримувачів (як користувачів сервісу так і поштові адреси)
5. Тема повідомлення
6. Текст повідомлення
7. Кнопка видалення повідомлення

Якщо дата відправки ще не настала контентий блок приймає вигляд, який повністю співпадає з виглядом форми створення листа за виключенням що поля ініціалізовані значеннями як в повідомленні, що з введеним <uuid:message_url>, і під кнопою підтвердження є кнопка Delete яка видаляє повідомлення і відміняє його відправку.

3.1.3. Опис форм

Для реєстрації та входу були використані в Django форми, однак для їх відображення в інтерфейсі сторінки були написані власні шаблони, які використовують вбудовані в поля фунуціїдля перевірки валідації.

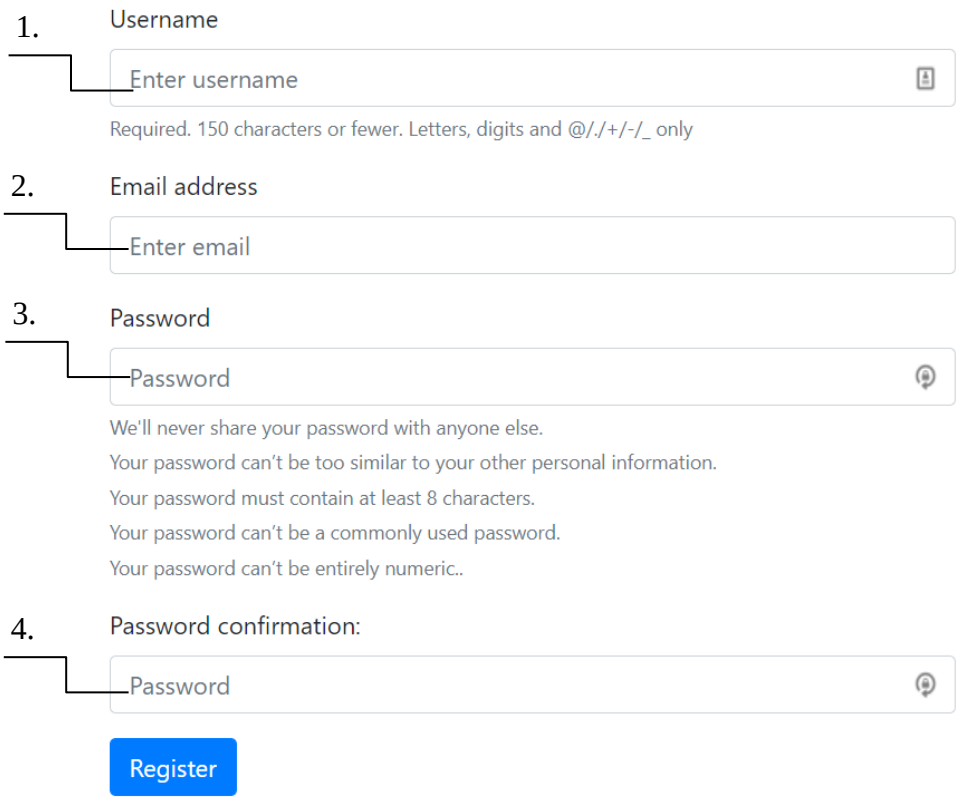


Рис. 3.5 Шаблон для реєстрації.

Складаєься з таких полів:

1. логін
2. електронна адреса
3. пароль
4. підтвердження паролю

Для реєстрації користувача потрібно заповнити всі поля форми:

Якщо поле заповнене неправильно з'явиться повідомлення про помилку, перелік помилок представлений нижче:

1. This field is required. – хоча б одне з полів не заповнене.
2. Enter a valid email address. – введений неправильний формат електронної адреси.
3. The password is too similar to the email address. – пароль схожий з електронною адресою.
4. This password is too short. It must contain at least 8 characters. – пароль занадто короткий.

5. This password is too common. – пароль занадто розповсюджений, простий.
6. This password is entirely numeric. – пароль складається лише з чисел.
7. The two password fields didn’t match. – пароль різниться з підтвердженням паролю.
8. This name cannot be registered. Please choose a different name. – логін має недопустимі символи

Для авторизації користувачів використовується шаблон наведений нижче.

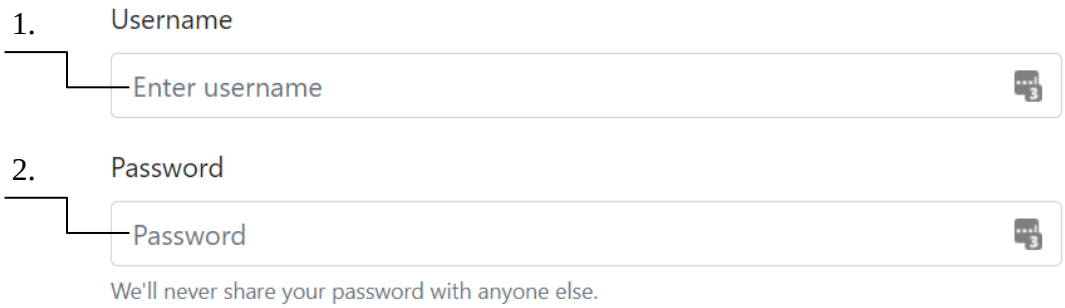


Рис. 3.6 Шаблон для авторизації.

Складаєься з таких полів:

1. логін
2. пароль

Для авторизації користувача потрібно заповнити всі поля.

Помилки:

1. Please enter a correct username and password. Note that both fields may be case-sensitive. – немає користувачів з введеним логіном та паролем

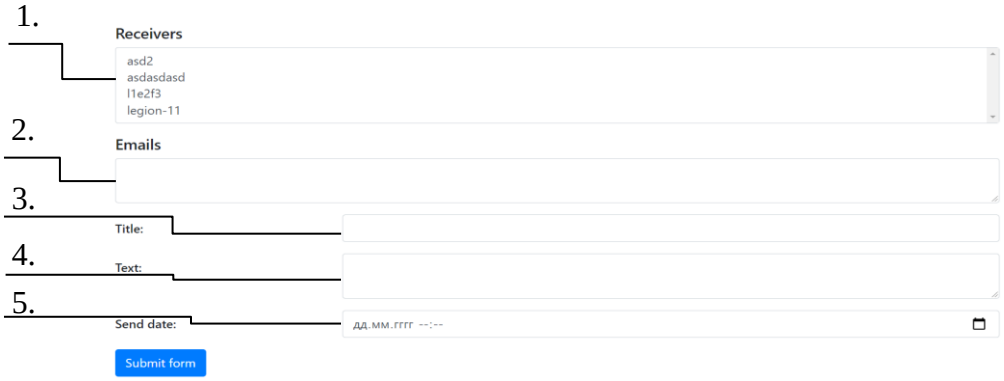


Рис. 3.7 Шаблон для створення повідомлень.

Складаєься з таких полів:

1. Перелік зареєстрованих користувачів сервісу, які отримають повідомлення.
2. Перелік електронних адрес осіб, які отримають повідомлення.
3. Заголовок повідомлення.
4. Основний текст повідомлення.
5. Дата та час коли повідомлення буде надіслано отримувачам.

Для створення нового повідомлення потрібно ввести логін користувача або електронну адресу, всі інші поля є опціональними.

Помилки:

1. No receivers or emails – не введено ні логін користувача, ні електронні адреси.
2. Invalid datetime – введена дата та час раніша за поточну дату та час.

Форма для створення повідомлень також використовується для сторінок редагування невідправлених повідомлень та сторінки створння повідомлень з шаблону.

3.2. Розробка БД

Проек Django надає власно створені моделі для Користувачів (User), груп користувачів (Group) їх прав (Permissions), перевірки автентифікації (LogEntry) і параметрів сесії (Sessions).

На основі вже існуючих моделей можна розробити модель нашої бази даних. Додавши такі моделі як: Message, Receivers. Далі буде надано опис полів моделей.

Message – модель для репрезентації повідомлень в базі даних.

```python
class Message(models.Model):
    sender = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, related_name='sender')
    emails = MultiEmailField(blank=True, null=True)
    url = models.UUIDField(default=uuid4, editable=False)
    title = models.CharField(max_length=200, blank=True)
    text = models.TextField(blank=True, null=True)
    send_date = models.DateTimeField('date to send', blank=True, null=True)
    pub_date = models.DateTimeField('date message created', auto_now_add=True)
    show = models.BooleanField(default=True)
    show_template = models.BooleanField(default=False)

    def __str__(self):
        return str(self.title)

    class Meta:
        verbose_name = 'Message'
        verbose_name_plural = 'Messages'
```

Рис. 3.8 Модель для репрезентації повідомлень в БД.

- id – ідентифікатор екземпляру моделі.
- sender – поєднує юзера який створив лист з повідомленням.
- emails – зберігає поштові адреси отримувачів
- pub_date – час створення повідомлення, заповнюється автоматично.
- send_date – час в який буде відправлено повідомлення.
- show – показує чи потрібно відображати лист в списку відправлених.
- show_template – показує чи потрібно відображати лист в списку шаблонів.
- text – текст повідомлення.
- title – заголовок повідомлення.
- url – uuid-код який буде вважатись посиланням на повідомлення.

```python
class Receivers(models.Model):
    user = models.ForeignKey(User, blank=True, default=None, related_name='user', on_delete=models.CASCADE)
    read = models.BooleanField(default=False)
    message = models.ForeignKey(Message, blank=True, default=None, related_name='receivers', on_delete=models.CASCADE)
    show = models.BooleanField(default=True)

    def __str__(self):
        return "{} {}".format(str(self.id), str(self.user))
```

Рис. 3.9 Модель для репрезентації відношення повідомлення-отримувач.

- id – ідентифікатор екземпляру моделі.
- message – поєднує юзера який отримає лист з повідомленням.
- user – поєднує юзера який отримає лист з повідомленням.
- read – показує чи був лист прочитаний.
- show – показує чи потрібно відображати лист в списку отриманих.

Важливо додати що для додачі моделей до бази даних потрібно спочатку підключити базу даних в файлі setings.py, встановити модуль django-multi-

email-field, а також виконати команду python manage.py makemigrations і команду python manage.py migrate в папці проекту де знаходиться файл manage.py.

3.3. Реалізація відправки повідомлення.

Для реалізації відправки повідомлень на поштові адреси в момент коли настає час відправки було використано модуль APScheduler [14]. Він надає функції планувальника, плануючи роботу, потрібно вибрати тригер. Тригер визначає логіку, за якою обчислюються дати / час, коли буде виконуватися завдання. APScheduler поставляється з трьома вбудованими типами тригерів:

date: використовується, коли ви хочете виконати роботу лише один раз у певний момент часу.

interval: використовується, коли ви хочете виконувати завдання через встановлені інтервали часу.

cron: використовується, коли ви хочете періодично виконувати завдання протягом певного часу.

в поєднанні з бібліотекою django.core.mail та тригера date були створені функції

```
def just_send_mail(message):
    send_mail(message.title, message.text, message.sender, message.emails, fail_silently=False)


def send_mail_sheduled(message):
    if message.send_date:
        scheduler.pause()
        scheduler.add_job(func=just_send_mail, trigger='date',
                          run_date=message.send_date, args=[message], id=str(message.url))
        scheduler.resume()
    else:
        just_send_mail(message)


def del_shcedule(message):
    try:
        scheduler.remove_job(str(message.url))
    except:  # JobLookupError
        print('not found')
```

Рис. 3.10 Функції для вирішення задачі відправки повідомлень

just_send_mail – відправляє повідомлення.

send_mail_sheduled – визначає чи потріюно запланувати відправку повідомлення, чи відправити негайно.

del_shcedule – видаляє задачу відправки повідомлення з черги планувальника, якщо задача ще не була виконана.

Задача відправки повідомлення користувачам сервісу не потребує задії планувальника, повідомлення, які ще не відправлені просто не відображаються в списку отриманих повідомлень, а при переході на сторінку read_message/<uuid:message_url> з їх полем url відображається пуста сторінка.

Висновки до третього розділу

У цьому розділі було наведено список необхідних інструментів та бібліотек та проведено огляд базової архітектури проектів на Django. Наведено перелік всіх url-ів проекту, та пояснено за які сторінки вони відповідають. Наведено опис всіх елементів інтерфейсу сторінок та елементів форм для реєстрації, авторизації, створення повідомлення.

До кожної з вище описаних форм було надано список відстежуваних помилок та пояснено, що їх викликає. Наостанок, було приведено опис моделей, за якими буде сформована модель бази даних, та надано обґрунтування використання того чи іншого поля моделі.

Надана в розділі інформація показує, що даного функціоналу повинно бути достатньо для вирішення поставлених функціональних завдань дипломної роботи, а саме реалізовано методи, які проводять розсилку повідомлень в заданий час.

РОЗДІЛ 4.

ТЕСТУВАННЯ

Тестування програмного забезпечення - визначається як діяльність щодо перевірки відповідності фактичних результатів роботи програми, до очікуваних результатів та забезпечення того, щоб програмна система була без дефектів.

4.1. Тестування реєстрації та авторизації.

В даному розділі проведена демонстрація роботи сервісу, та поетапно буде показана робота всієї системи. Перш за все потрібно запустити додаток за допомогою команди python manage.py runserver, за замовчуванням будемо вважати, що всі необхідні дії, такі як створення міграцій, та базове налаштування уже виконане. Тестування буде проводитись на локальному пристрої, тому додаток буде відкриватись в браузері за посиланням http://127.0.0.1:8000/ (порт 8000 обирається в додатках Django як за замовчуванням)

Перше, що потрібно зробити користувачу при роботі з додатком при роботі з сервісом це зареєструватись.

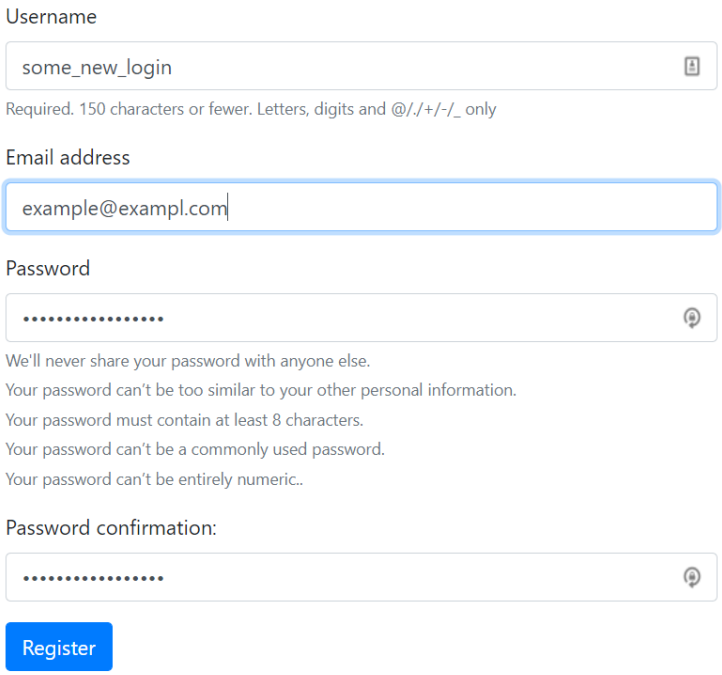


Рис. 4.1 Приклад заповненої форми реєстрації

Для прикладу спробуємо зареєструвати нового користувача з логіном some_new_login та паролем some_new_password.

Як можна побачити у вікні адміністратора новий акаунт користувача був успішно створений.

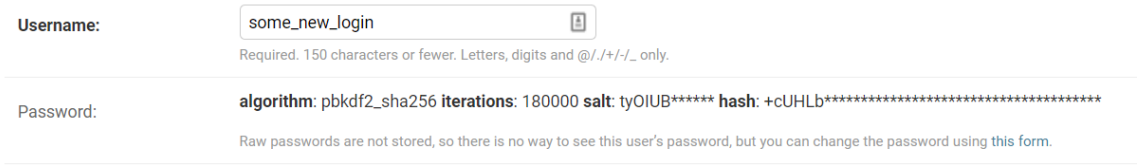


Рис. 4.2 Збережені дані користувача

Як ми можемо побачити пароль користувача не співпадає з введеним паролем, адже база даних не зберігає його у чистому вигляді, а зберігає лише хеш-код паролю з сіллю, для хешування використовується хеш алгоритм sha256. Таким чином персональний пароль користувача захщений від користувачів з правами адміністратора.

Після того як реєстрація пройшла успішно, користувач буде автоматично авторизований і відбудеться пересилання на сторінку http://127.0.0.1:8000/received/0.

Однак з ціллю тестування ми вийдемо з акаунту користувача нажавши кнопку logout в правому верхньому кутку та спробуємо повторно авторизуватись.

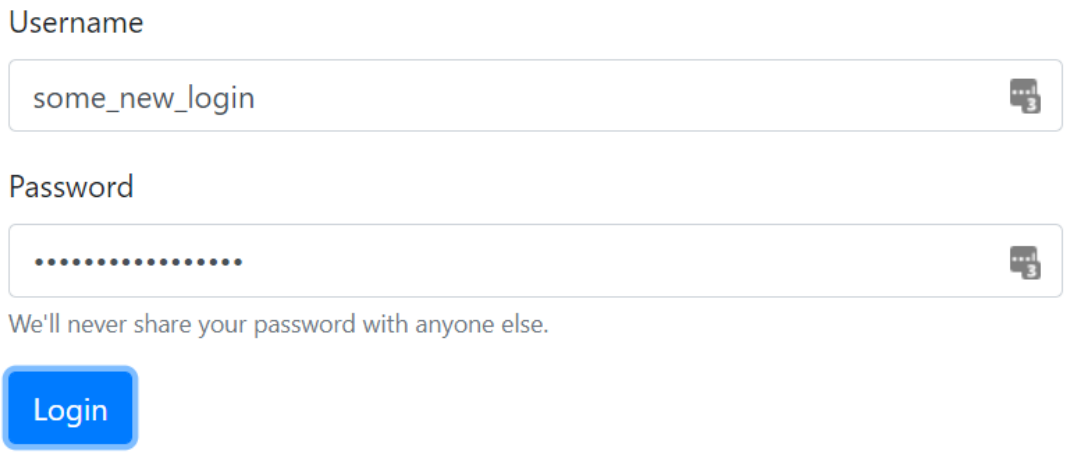


Рис. 4.3 Приклад заповненої форми авторизації

Як можна побачити додаток успішно авторизує користувача, а отже введений пароль формує ідентичний збереженому хеш-код.

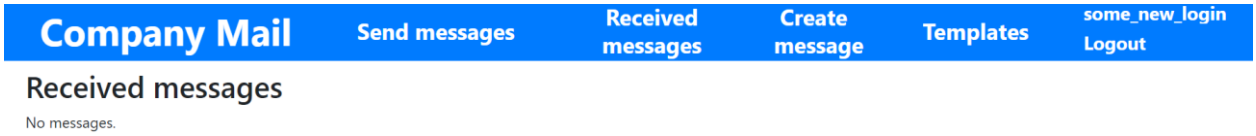


Рис. 4.4 Контент сторінки http://127.0.0.1:8000/received/0 після авторизації

4.2. Тестування відправки повідомлення до користувачів сервісу.

Щоб переконатись, що все працює, спробуємо надіслати повідомлення одному з користувачів сервісу. Для цього перейдемо на сторінку http://127.0.0.1:8000/new за допомогою кнопки Create message на основній панелі. Перейшовши на відповідну сторінку заповнимо форму потрібними нам данними, в даному прикладі, повідомлення буде відправленно до користувача з логіном legion-11 і поточному користувачу (збережемо як шаблон для подальшого тестування).

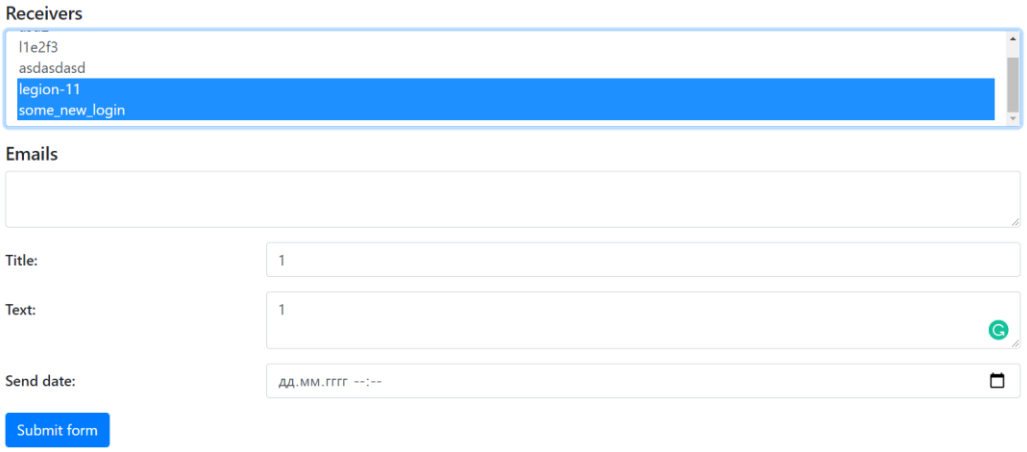


Рис. 4.5 Приклад заповнених полів форми.

Після успішного створення листа, відбудеться пересилання на сторінку http://127.0.0.1:8000/send/0

Перед тим як перевірити вкладку БД на предмет наявності новоствореного листа, створимо що один лист, однак на цей раз також заповнимо поле Send date, датою 04.06.2020 20:20. інші попередьо введені поля замінемо на двійки.

Наведена ілюстрація нижче підтверджує повну працездатність функціоналу зі створення повідомлень та правильного відображення відправлених та отриманих повідомлень.

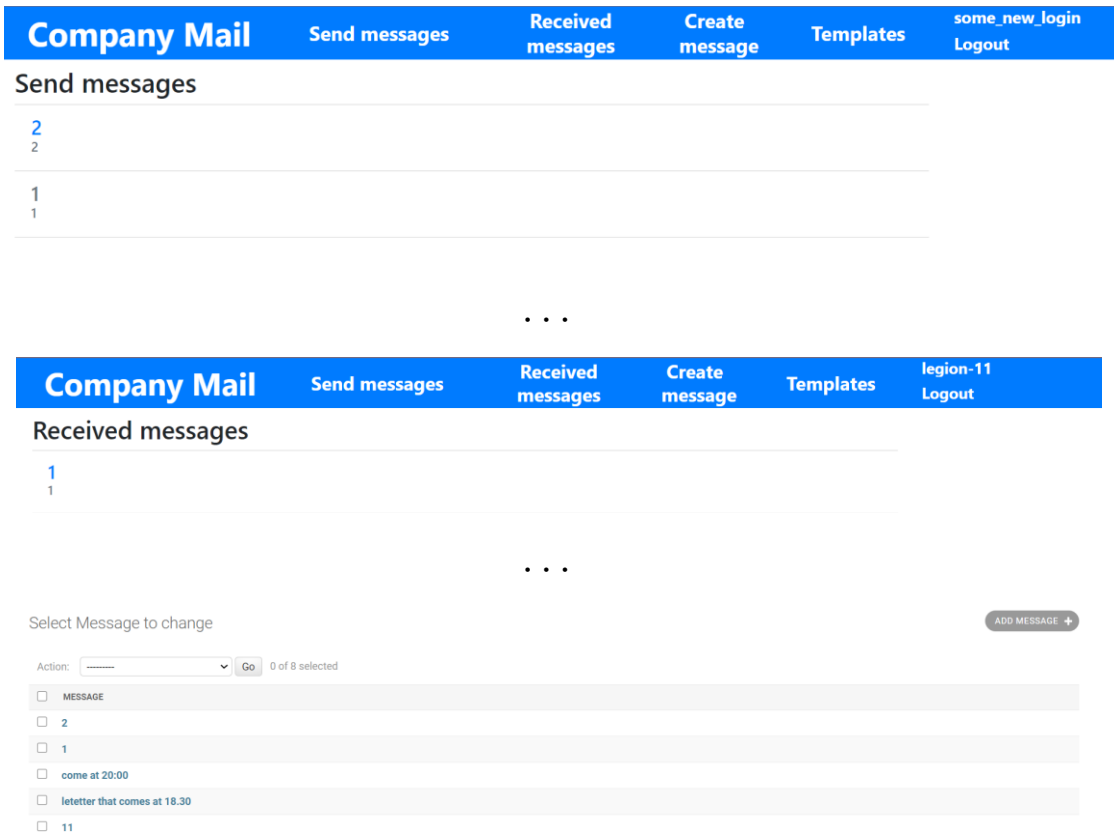


Рис. 4.6 Список відправлених повідомлень в інтерфейсі користувача - відправника, список отриманих повідомлень в інтерфейсі користувача отримувача, та список створених повідомлень у інтерфейсі адміністратора.

Необхідно перевірити відображення даних повідомлень у інтерфейсі користувача, і можливість редагування та видалення повідомлень.

Як можна побачити нижче інтерфейси для надісланих і не надісланих повідомлень сильно відрізняються. Так повідомлення сліва (надіслане), представлене звичайним відредагованим текстом в той час, як не надіслане представлене формою, яка надає можливість редагувати повідомлення, або його повністю видалити (відмінити надсилання).

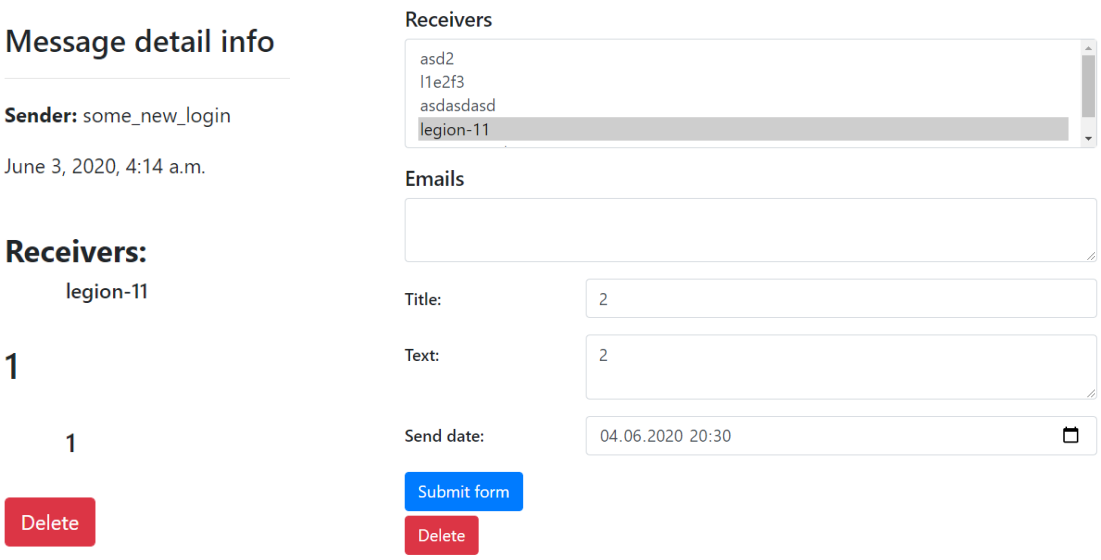


Рис. 4.7 Відображення надісланого та не надісланого повідомлення в інтерфейсі користувача-відправника.

Після того як не надіслане повідомлення було видалене воно повністю зникає з бази даних, на відміну від надісланого, яке для повного видалення також повинне бути видалене отримувачем і зі списку шаблонів, що продемонстровано на рисунку нижче.

Рис. 4.8 Список створених повідомлень у інтерфейсі адміністратора, після видалення не надісланого повідомлення.

Наостанок перевіримо створення нового повідомлення зі збереженого шаблону. Як можна побачити з ілюстрації нижче нове повідомлення повністю при виборі єдиного повідомлення зі списку шаблонів автоматично створюється форма, як при створенні нового листа за виключенням, що поля ініціалізовані даними з повідомлення-шаблону.

Рис. 4.9 Створення нового повідомлення з шаблону повідомлення.

На цьому всі тестування відправки повідомлення до користувачів сервісу, можна вважати завершеним, і можна достовірно вважати, що система виконує задуманий функціонал.

4.3. Тестування відправки повідомлення за поштовим адресом.

Для тестування відправки повідомлень на введені поштові адреси було вирішено використати mailtrap. Mailtrap[15] - це несправжній SMTP-сервер, призначений для тестування, перегляду та обміну електронними листами, що надсилаються з середовищ розробки та інсценування без спаму справжніх клієнтів або затоплення власної поштової скриньки. Для використання Mailtrap на сайті було створено новий акаунт, а у файлі settings.py було ініціалізовані такі змінні як EMAIL_HOST, EMAIL_HOST_USER, EMAIL_HOST_PASSWORD, EMAIL_PORT в значення надані акаунтом.

Для тестування функції надсилання поштового повідомлення були створені два повідомлення, Перше повинне бути надіслане за адресами example@example.com, example2@example.com, у момент створення (тобто 03.06.2020 5.52), а друге повинне бути надіслане за адресами example3@example.com, example4@example.com, у момент 6.00.

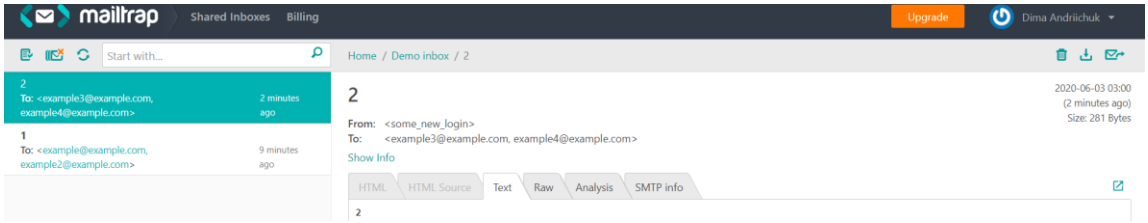


Рис. 4.10 демонстрація відправки поштових повідомлень за вказаним часом.

Як видно з рисунку нижче повідомлення були надіслані точно у вказані моменти, за різницею, що Mailtrap показує час відносно нульового часового поясу, тобто показані на рисунку 3 години ранку, це і 6 годин ранку за Київським часом.

Даного тесту повинно бути цілком достатньо для перевірки відправки повідомлень на поштову адресу, так як всі інші випадки тестування, такі як редагування та видалення повідомлень повністю співпадають з відповідіними тестуваннями відправки повідомлення до користувачів сервісу. і уже були оглянуті в попередньому підрозділі.

Висновки до четвертого розділу

В даному розділі було продемонстровано роботу розробленого сервісу для відкладеної відправки поштових повідомлень. Демонстрація включає в себе почергові інструкції для користувача, та супроводиться необхідні для розуміння знімками екрану, що дозволяє розібратися зі всіма елементами інтерфейсу системи.

Сервіс має простий та інтуїтивно зрозумілий інтерфейс користувача, що надає всю необхідну інформацію для роботи в системі. Розроблена програма, повністю відповідає функціональним вимогам наведеним в розділі 2.

Повна працезданіть програми показує про правельність прийнятих рішень на етапі вибору бібліотек та проектування.

ВИСНОВКИ

Метою даної дипломної роботи було створння сервісу для відкладеної відправки поштових повідомлень. Під час її розробки було торетично опрацьовно та досліджено велика кількість матеріалів, та інструментів для реалізації поставленої задачі. На основі прийнятих рішень було отримано наступні висновки.

Ефективна комунікація - важливий елемент для досягнення бізнес-цілей. Будь-якій компанії необхідно бути на зв'язку зі своїми клієнтами, а технології надають нам безліч варіантів зв'язку з цим.І хоч за останні роки розвиток месенджерів дійсно вражає (навіть смс розсилка починає втрачати обороти, не витримуючи конкуренції), поштова розсилка як і раніше знаходиться на першому місці в світі бізнес-комунікацій. Тільки у Gmail більше 1 мільярда користувачів по всьому світу. А посягає вона перше місце завдяки тому, що:

- Зазвичай у всіх, з ким ви працюєте або спілкуєтеся, є адреса електронної пошти. Він універсальний - ви можете відправити лист кому завгодно. Наприклад, Viber розсилка або будь-яка інша відправляє повідомлення тільки клієнтам з адресної книги.
- Одержувачі можуть відповідати на листи в зручному для них темпі.
- Вона зберігає історію, на відміну від sms розсилки. Листи можуть бути централізовано збережені або архівовані.

І в цілому після популяризації месенджерів, поштові сервіси також почали покращувати та розширяти вже існуючий функціонал.

Однак як було сказано в першому розділі дипломної роботи, ніщо не може гарантувати того, що компанія яка надає послугу надсилання поштових повідомлень не збирає дані повідомлень у власних цілях, або того що сервіс буде дескридетовано працівниками, і тим самим втрачено таємницю листування.

З цієї причини з'являється багато корпоративних мобільних платформ для обміну повідомленнями. Вони забезпечують адміністрування листів і їх безпеку. Однією з таких корпоративних платформ і являє собою реалізований мною сервіс для відкладеної відправки поштових повідомлень.

В дипломі було чітко пояснено доцільність використання такого сервісу для обміну повідомленнями всередині компанії, так і при проводженні массової розсилки від імені компанії. І хоч наведена реалізація являє собою лише мінімальний базис для виконання функціональних завдань наведеного прикладу, однак система написана у архітектурі яка підлягає масштабуванню, а отже і розширенню функціоналу для задіювання в більш серйозних та перспективних проектах.

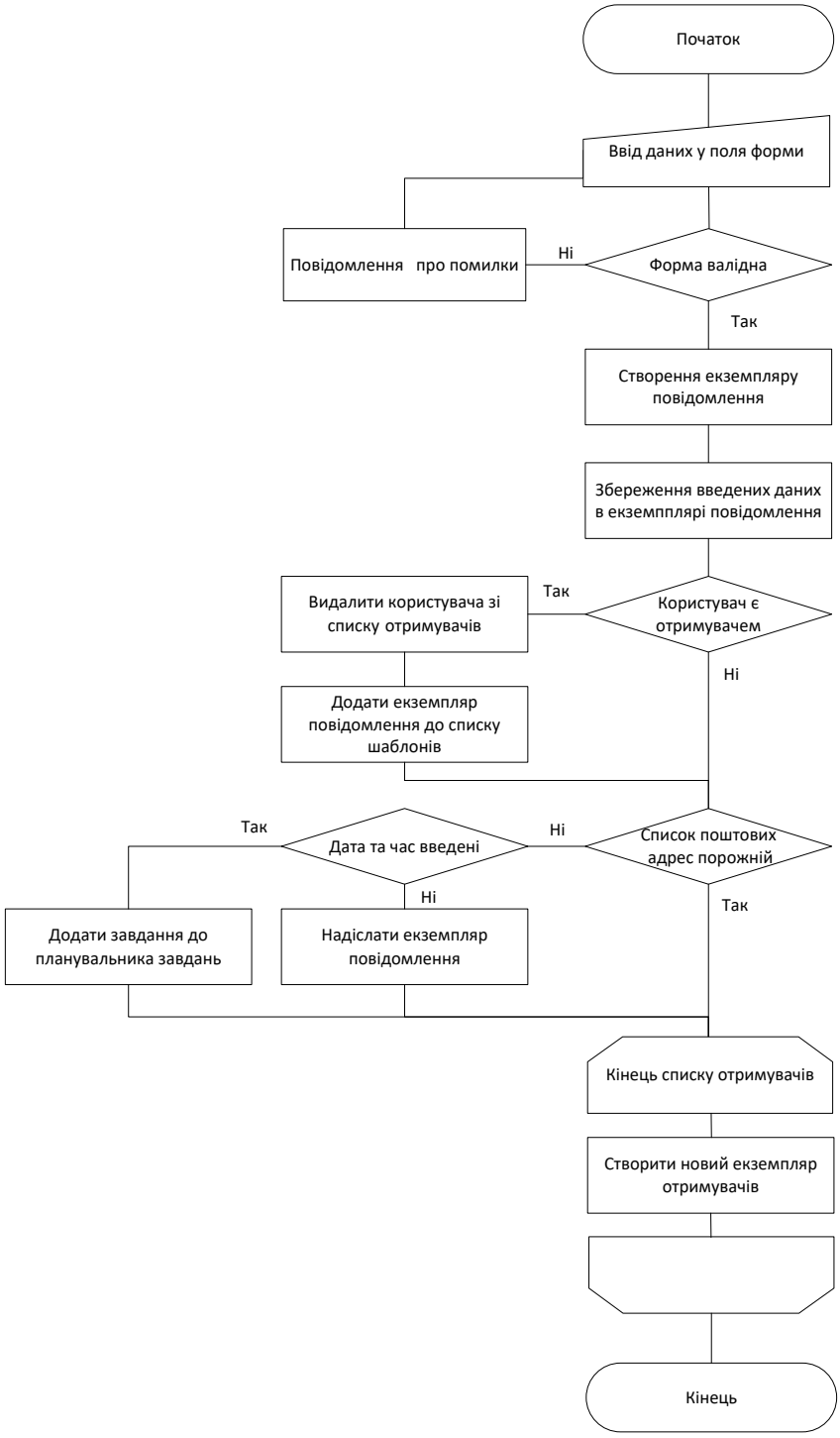
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Что такое SMTP, как работает электронная почта. [Електронний ресурс] – Режим доступу до ресурсу: https://imacros.ru/raznoe/kak-rabotaet-pochtovyj-server.html.
2. Рекомендации по организации архитектуры почтовой системы домена и ее интеграция с сервисами RBL/DNSBL [Електронний ресурс] – Режим доступу до ресурсу: http://rbldns.ru/index.php/ru/arch-faq.html..
3. Developing an Instant Messaging Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/cd/E19142-01/819-0063/im-architecture.html.
4. de la Guardia C. Python Web Frameworks / Carlos de la Guardia., 2016.
5. Dauzon S. Django: Web Development with Python / S. Dauzon, A. Bendoraitis, A. Ravindran., 2016
6. Морето С. Bootstrap в примерах / Сильвио Морето., 2016. – 314 с.
7. Создание рабочей базы данных [Електронний ресурс] – Режим доступу до ресурсу: https://djbook.ru/rel3.0/topics/install.html#database-installation.
8. SQLite on Heroku [Електронний ресурс] – Режим доступу до ресурсу: https://devcenter.heroku.com/articles/sqlite3.
9. Heroku Dynos [Електронний ресурс] – Режим доступу до ресурсу: https://www.heroku.com/dynos.
10. https://djbook.ru/ch05s02.html [Електронний ресурс] – Режим доступу до ресурсу: Методика MTV (или MVC).
11. Разделение визуализации и бизнес-логики [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikiversity.org/wiki/Разделение_визуализации_и_бизнес-логики.
12. Радужные таблицы в домашних условиях [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/145820.

13. UUID objects according to RFC 4122 [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/library/uuid.html.
14. Advanced Python Scheduler. [Електронний ресурс] – Режим доступу до ресурсу: https://apscheduler.readthedocs.io/en/ stable/index.html.
15. General Questions about Mailtrap [Електронний ресурс] – Режим доступу до ресурсу: https://mailtrap.io/faq#faq01.

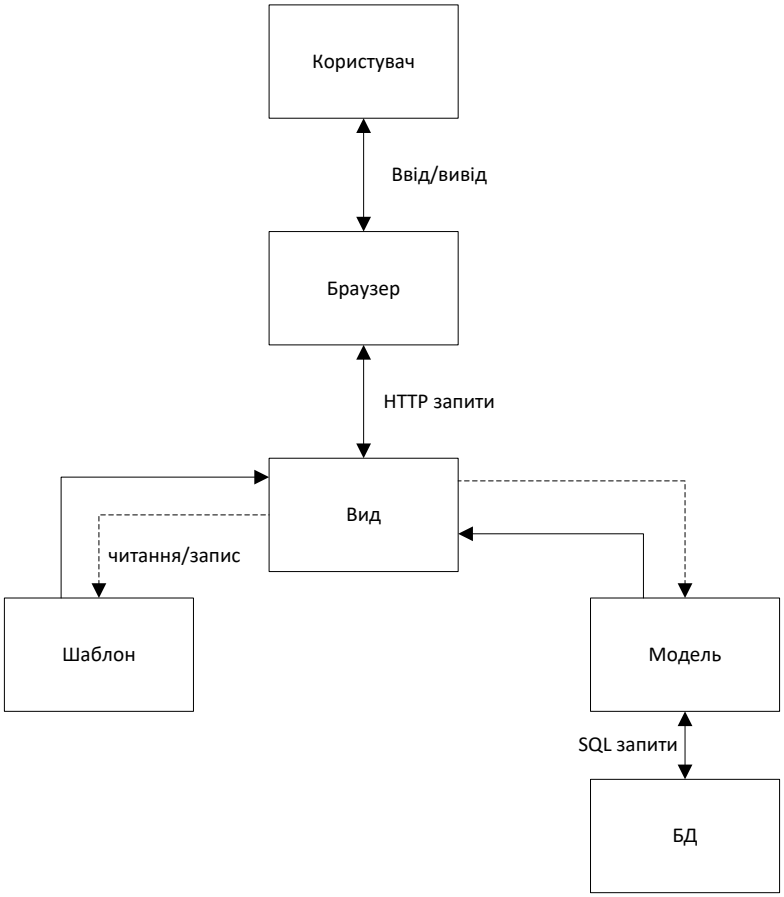
ДОДАТКИ

Схема Алгоритму



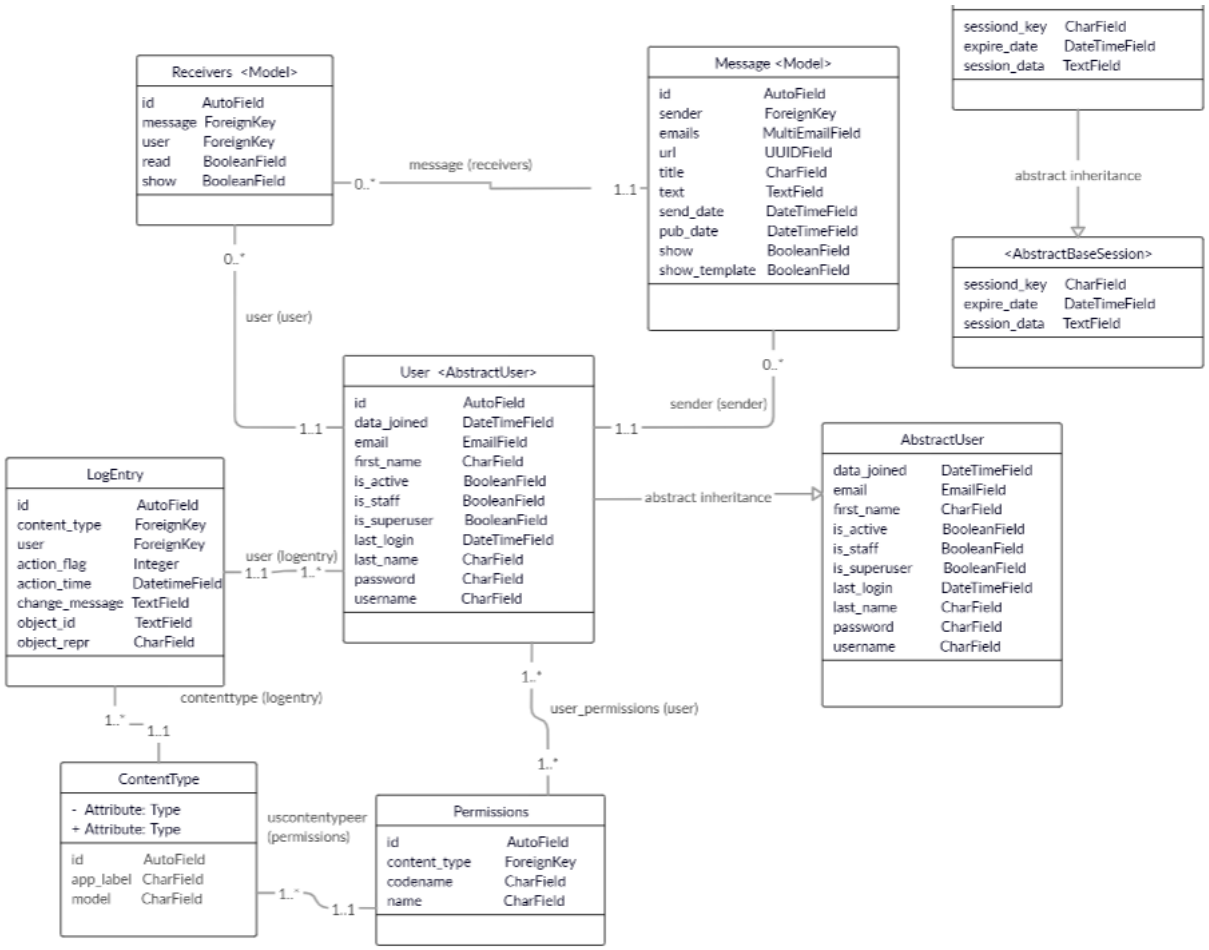
					ДП.6401.04.000 Д1		
Зм.		*№ документа*	*Підп.*	*Дата*			
Розробив		*Анлрійчук Д.А.*			*Сервіс для відкладеної відправки поштових повідомлень*	*Літ.*	*Аркуш*
Перевір.		*Каплунов А. В.*				Т	2 / 1
Н.конт		*Симоненко В.П.*			*Додаток А*	*НТУУ «КПІ імені Ігора Сікорського», ФІОТ Група ІО - 64*	
Затв.							

Функціональна Схема



Зм.		№ документа	Підп.	Дата	ДП.6401.05.000 Д2		
Розробив		Андрійчук Д.А.			Сервіс для відкладеної відправки поштових повідомлень Додаток Б	Літ.	Аркуш 1 / 1
Перевір.		Симоненко В.П.				Т	
Н.конт.		Каплунов А. В.				НТУУ «КПІ імені Ігора Сікорського», ФІОТ Група ІО - 64	
Затв.							

Діграма Класів



					ДП.6401.06.000 Д3		
Зм.		№ документа	Підп.	Дата			
Розробив		Андрійчук Д.А.			Сервіс для відкладеної відправки поштових повідомлень Додаток В	Літ.	Аркуш
Перевір.		Каплунов А. В.				Т	1 / 1
Н.конт		Симоненко В.П.				НТУУ «КПІ імені Ігора Сікорського», ФІОТ Група ІО - 64	
Затв.							

Лістінг Програми

mysite/settings.py

```
import os
from decouple import config

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = config('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    # 'mymail.apps.MymailConfig',
    'mymail',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'multi_email_field',
    'django_extensions',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

```
ROOT_URLCONF = 'mysite.urls'


TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, r'mymail\templates\mymail')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'mysite.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST'),
        'PORT': '5432',
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'

EMAIL_HOST = config('EMAIL_HOST')
EMAIL_HOST_USER = config('EMAIL_HOST_USER')
EMAIL_HOST_PASSWORD = config('EMAIL_HOST_PASSWORD')
EMAIL_PORT = config('EMAIL_PORT')
EMAIL_USE_TLS = True

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

mysite/urls.py

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('accounts/', include('django_registration.backends.activation.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
    path('admin/', admin.site.urls),
    path('', include('mymail.urls')),
]
```

mysite/mymail/admin.py

```
from django.contrib import admin
from .models import Message, Receivers, Metadata

admin.site.register(Message)
admin.site.register(Receivers)
admin.site.register(Metadata)
```

mysite/mymail/forms.py

```
from django import forms
from .models import User, Message, Metadata
import django.utils.timezone as timezone
```

```
from multi_email_field.forms import MultiEmailField
from django.utils.translation import gettext as _


class DateTimeInput(forms.DateTimeInput):
    input_type = "datetime-local"

    def __init__(self, **kwargs):
        kwargs["format"] = "%Y-%m-%dT%H:%M"
        super().__init__(**kwargs)


class MessageForm(forms.ModelForm):
    receivers = forms.ModelMultipleChoiceField(User.objects, required=False,)
    title = forms.CharField(required=False)
    text = forms.Textarea()
    send_date = forms.DateTimeField(required=False, widget=DateTimeInput,
                                    input_formats=["%Y-%m-%dT%H:%M", "%Y-%m-%d %H:%M"])
    emails = MultiEmailField(required=False)
    # file_field = forms.FileField(required=False, widget=forms.ClearableFileInput(attrs={'multiple': True}))

    def __init__(self, *args, **kwargs):
        super(MessageForm, self).__init__(*args, **kwargs)
        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'form-control'
        self.fields["emails"].widget.attrs['rows'] = "2"
        self.fields["text"].widget.attrs['rows'] = "2"

    class Meta:
        model = Message
        fields = ['receivers', 'emails', 'title', 'text', 'send_date']

    def clean_send_date(self):
        date = self.cleaned_data.get('send_date')
        cd = self.cleaned_data
        if date:
            if date < timezone.now():
                raise forms.ValidationError(_('Invalid datetime: %(value)s'),
                      params={'value': date},
                  )
        return date

    def clean(self):
        receivers = self.cleaned_data.get('receivers')
        emails = self.cleaned_data.get('emails')
        if not receivers and not emails:
            raise forms.ValidationError(_('Error: No receivers or emails'))
        return self.cleaned_data
```

mysite/mymail/models.py

```
from django.db import models
from django.contrib.auth.models import User
from uuid import uuid4
from multi_email_field.fields import MultiEmailField
```

```
class Message(models.Model):
    sender = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, related_name='sender')
    emails = MultiEmailField(blank=True, null=True)
    url = models.UUIDField(default=uuid4, editable=False)
    title = models.CharField(max_length=200, blank=True)
    text = models.TextField(blank=True, null=True)
    send_date = models.DateTimeField('date to send', blank=True, null=True)
    pub_date = models.DateTimeField('date message created', auto_now_add=True)
    show = models.BooleanField(default=True)
    show_template = models.BooleanField(default=False)

    def __str__(self):
        return str(self.title)

    class Meta:
        verbose_name = 'Message'
        verbose_name_plural = 'Messages'


class Receivers(models.Model):
    user = models.ForeignKey(User, blank=True, default=None, related_name='user',
on_delete=models.CASCADE)
    read = models.BooleanField(default=False)
    message = models.ForeignKey(Message, blank=True, default=None, related_name='receivers',
on_delete=models.CASCADE)
    show = models.BooleanField(default=True)

    def __str__(self):
        return "{} {}".format(str(self.id), str(self.user))


class Metadata(models.Model):
    title = models.CharField(max_length=255, blank=True)
    file = models.FileField(upload_to=r'uploads/%Y/%m/%d/', blank=True)
    message = models.ForeignKey(Message, on_delete=models.CASCADE, null=True, related_name='meta')

    class Meta:
        verbose_name = 'Metadata'
        verbose_name_plural = 'Metadata'
```

mysite/mymail/urls.py

```
from django.urls import path
from django.contrib.auth.views import LogoutView, LoginView
from django_registration.backends.one_step.views import RegistrationView
from . import views


urlpatterns = [
    path('', views.redir, name='home'),
    path('received/<int:page>', views.search_received, name='search_received'),
    path('send/<int:page>', views.search_send, name='search_send'),
    path('templates/<int:page>', views.search_templates, name='search_templates'),
    path('<uuid:message_url>', views.read_message, name='read_message'),
    path('new', views.create_message, name='create_message'),
```

```
    path('delete/<uuid:message_url>', views.delete, name='delete'),
    path('just_delete/<uuid:message_url>', views.just_delete, name='just_delete'),
    path('delete/<uuid:message_url>/Template', views.delete_template, name='delete_template'),
    path('new/<uuid:message_url>', views.create_from_template, name='create_from_template'),
    path(r'registration/',
        RegistrationView.as_view(template_name="mymail/registration.html", success_url="/"),
        name='registration'),
    path(r'logout/', LogoutView.as_view(), name='logout'),
    path(r'login/', LoginView.as_view(template_name="mymail/login.html"), name='login_view'),
]
```

mysite/mymail/views.py

```
from django.shortcuts import render
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login
from django.shortcuts import redirect, get_object_or_404
from .forms import MessageForm
from .models import Message, Receivers
import datetime
from apscheduler.schedulers.background import BackgroundScheduler
from django.utils.timezone import activate, now
from django.core.exceptions import ObjectDoesNotExist
from django.core.mail import send_mail
scheduler = BackgroundScheduler()
scheduler.start()


def just_send_mail(message):
    send_mail(message.title, message.text, message.sender, message.emails, fail_silently=False)


def send_mail_sheduled(message):
    if message.send_date:
        scheduler.pause()
        scheduler.add_job(func=just_send_mail, trigger='date',
                          run_date=message.send_date, args=[message], id=str(message.url))
        scheduler.resume()
    else:
        just_send_mail(message)


def del_shcedule(message):
    try:
        scheduler.remove_job(str(message.url))
    except:  # JobLookupError
        print('not found')


def search(request, page, messages, what_search):
    number_print = 30
    more = len(messages) > number_print * (page + 1)
    return render(request, 'mymail/search.html',
                  {"messages": messages[number_print * page: number_print * (page + 1)],
                   "page": page,
                   "more": more, "what_search": what_search})
```

```
def search_received(request, page):
    activate('Europe/Kiev')
    if request.user.is_authenticated:
        current_user = request.user
        messages = Receivers.objects.filter(show=True, user=current_user)\
            .exclude(message__send_date__isnull=False,
                     message__send_date__gt=datetime.datetime.now())\
            .order_by('-message')
        return search(request, page, messages, what_search='Received messages')
    else:
        return render(request, 'mymail/search.html', {'what_search': 'Received messages'})


def search_send(request, page):
    activate('Europe/Kiev')
    if request.user.is_authenticated:
        current_user = request.user
        messages = Message.objects.filter(sender=current_user, show=True).order_by('-id')
        return search(request, page, messages, what_search='Send messages')
    else:
        return render(request, 'mymail/search.html', {'what_search': 'Send messages'})


def read_message(request, message_url):
    activate('Europe/Kiev')
    current_user = request.user
    try:
        message = Message.objects.get(url=str(message_url))
        receiver = message.receivers.filter(user=current_user)
        if current_user == message.sender:
            if message.send_date and message.send_date > now():
                initial_dict = {
                    "title": message.title,
                    "receivers": [r.user for r in message.receivers.all()],
                    "text": message.text,
                    "send_date": message.send_date,
                    "emails": message.emails
                }
                form = MessageForm(request.POST or None, initial=initial_dict)
                if request.method == "POST" and form.is_valid():
                    data = form.cleaned_data
                    message.title = data.get('title')
                    message.text = data.get('text')
                    message.send_date = data.get('send_date')
                    receivers_user = data.get('receivers')

                    to_delete = list(message.receivers.all().exclude(user__in=receivers_user))
                    ids=[r.user.id for r in message.receivers.all()]
                    to_add = list(receivers_user.exclude(id=current_user.id).exclude(id__in=ids))

                    for i in range(min(len(to_delete), len(to_add))):
                        r = to_delete.pop()
                        r.user = to_add.pop()
                        r.save()
```

```
                for r in to_delete:
                    r.delete()
                for u in to_add:
                    r = Receivers(user=u, message=message)
                    r.save()

                if message.emails:
                    del_shcedule(message)
                message.emails = data.get('emails')
                if current_user in receivers_user:
                    message.show_template = True
                message.save()

                if message.emails:
                    send_mail_sheduled(message)

                return redirect('send/0')
            return render(request, 'mymail/edit.html', locals())

        elif receiver:
            if not message.send_date or message.send_date < now():
                receiver = receiver.get()
                receiver.read = True
                receiver.save()
            else:
                message = ''
        else:
            message = ''
    except ObjectDoesNotExist:
        message = 'ObjectDoesNotExist'
        print('__ERROR__')
    return render(
        request,
        'mymail/read_message.html', {"message": message, "current_user": current_user}
    )


def redir(request):
    return redirect('received/0')


def search_templates(request, page):
    activate('Europe/Kiev')
    if request.user.is_authenticated:
        current_user = request.user
        messages = Message.objects.filter(sender=current_user, show_template=True).order_by('-id')
        return search(request, page, messages, 'Templates')
    else:
        return render(request, 'mymail/search.html', {'what_search': 'Templates'})


def delete(request, message_url):
    message = get_object_or_404(Message, url=message_url)
    user = request.user
    if user.is_authenticated:
        if message.sender == user:
```

```
            message.show = False
            message.save()
        elif user in [r.user for r in message.receivers.all()]:
            r = message.receivers.get(user=user)
            r.show = False
            r.save()
        delete_mess_without_show(message)
    return redirect('search_received', page=0)


def delete_template(request, message_url):
    message = get_object_or_404(Message, url=message_url)
    user = request.user
    if user.is_authenticated and message.sender == user:
        message.show_template = False
        message.save()
        delete_mess_without_show(message)
    return redirect('search_templates', page=0)


def delete_mess_without_show(message):
    if message.show is False and not message.receivers.filter(show=True) and not message.show_template:
        del_shcedule(message)
        message.delete()


def just_delete(request, message_url):
    message = get_object_or_404(Message, url=message_url)
    if request.user == message.sender and message.send_date > now():
        del_shcedule(message)
        message.delete()
    return redirect('home')


def creation(request, form, template=False, message=''):
    if request.method == "POST" and form.is_valid():
        print("request - {}".format(request.POST))
        data = form.cleaned_data
        print('data', data)
        receivers = data.get('receivers')
        title = data.get('title')
        text = data.get('text')
        send_date = data.get('send_date')
        emails = data.get('emails')

        message = Message(sender=request.user, title=title, text=text, send_date=send_date, emails=emails)
        if request.user in receivers:
            message.show_template = True

            if len(receivers) == 1:
                message.show = False
        message.save()
        if emails:
            send_mail_sheduled(message)
        for receiver in receivers.exclude(id=request.user.id):
            r = Receivers(user=receiver, message=message)
```

```
            r.save()
        return redirect('send/0')

    return render(request, "mymail/create_message.html", locals())


def create_from_template(request, message_url):
    activate('Europe/Kiev')
    try:
        message = Message.objects.get(url=message_url, sender=request.user)
        initial_dict = {
            "title": message.title,
            "receivers": [r.user for r in message.receivers.all().exclude(user=request.user)],
            "text": message.text,
            "send_date": message.send_date,
            "emails": message.emails
        }
    except ObjectDoesNotExist:
        return redirect('create_message')
    form = MessageForm(request.POST or None, initial=initial_dict)
    return creation(request, form, True, message)


def create_message(request):
    form = MessageForm(request.POST or None)
    return creation(request, form)
```

mysite/mymail/templatetags/mymail_extras.py

```
from django import template

register = template.Library()


@register.filter(name='add')
def add(value, arg):
    return value + arg


@register.filter(name='subtract')
def subtract(value, arg):
    return value - arg
```

mysite/mymail/templates/mymail/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

    <title>Company Mail</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">
</head>
<body>
```

```
<div class="container-fluid">
  {% include '_header.html' %}
</div>

{% block content %}
{% endblock %}


</body>
</html>
```

mysite/mymail/templates/mymail/_header.html

```
<div class="header row bg-primary">
  <div class="col-3 ml-0 text-center my-auto">
    <a class="text-decoration-none h1 text-white font-weight-bold" href="{% url 'home' %}">
      Company Mail
    </a>
  </div>

  <div class="col-2 text-center my-auto">
    <a class="text-decoration-none h4 text-white font-weight-bold" href="{% url 'search_send' 0 %}">
      Send messages
    </a>
  </div>

  <div class="col-2 text-center my-auto">
    <a class="text-decoration-none h4 text-white font-weight-bold" href="{% url 'search_received' 0 %}">
      Received messages
    </a>
  </div>

  <div class="col-1 text-center my-auto">
    <a class="text-decoration-none h4 text-white font-weight-bold" href="{% url 'create_message' %}">
      Create message
    </a>
  </div>

  <div class="col-2 text-center my-auto">
    <a class="text-decoration-none h4 text-white font-weight-bold" href="{% url 'search_templates' 0 %}">
      Templates
    </a>
  </div>

  <div class="col-2">
    {% if user.is_authenticated %}
      <div class="row font-weight-bold text-white h5 mr-2 text-truncate">
        {{ user.username }}
      </div>
      <div class="row">
        <a class="font-weight-bold text-decoration-none text-white h5"
          href="{% url 'logout' %}">
          Logout
        </a>
      </div>
```

```
    {% else %}
      <div class="row">
        <a class="font-weight-bold text-decoration-none text-white text-justify h5"
          href="{% url 'login_view' %}">
          Login
        </a>
      </div>
      <div class="row">
        <a class="font-weight-bold text-decoration-none text-white text-justify h5"
          href="{% url 'registration' %}">
          Registration
        </a>
      </div>
    {% endif %}

  </div>
</div>
```

mysite/mymail/templates/mymail/create_message.html

```
{% include "mymail/base.html" %}

{% block content %}
<div class="container mt-2 ml-2">

{% if user.is_authenticated %}
  <form method="post" action="{% url 'create_message' %}"> {% csrf_token %}
    <div class="form-group">
      <label class="h5 label" for="id_receivers">{{form.receivers.label}}</label>
      {{form.receivers}}
    </div>

    <div class="form-group ">
      <label class="h5 label" for="id_emails">{{form.emails.label}}</label>
      {{ form.emails }}
    </div>

    <div class="form-group row">
      <label class="h5 col-sm-3 col-form-label" for="id_title">{{form.title.label}}:</label>
      <div class="col-sm-9">
        {{ form.title }}
      </div>
    </div>

    <div class="form-group row">
      <label class="h5 col-sm-3 col-form-label" for="id_text">{{form.text.label}}:</label>
      <div class="col-sm-9">
        {{ form.text }}
      </div>
    </div>

    <div class="form-group row">
      <label class="h5 col-3 col-form-label" for="id_send_date">{{form.send_date.label}}:</label>
      <div class="col-9">
        {{form.send_date}}
      </div>
```

```
        </div>

        <button class="btn btn-primary" type="submit">Submit form</button>
        {%if form.errors%}
        <div class="alert alert-danger" role="alert">
          {{form.errors}}
        </div>
        {%endif%}
        {% if template %}
        <form action="{% url 'delete' message.url %}" method="post">
          {% csrf_token %}
        <input class="btn btn-danger" type="submit" value="Delete">
        {% endif %}

        <input type="hidden" name="next" value="{{ next }}"/>

      </form>

{% else %}
  <h1 class="ml-5 center text-center">You are not authenticated</h1>
{% endif %}

</div>
{% endblock %}
```

mysite/mymail/templates/mymail/edit.html

```
{% include "mymail/base.html" %}
{% block content %}
<div class="container ml-4 mt-2 col-6">

{% if user.is_authenticated %}
  <form action="{% url 'read_message' message.url %}" method="post"> {% csrf_token %}
    <div class="form-group">
      <label class="h5 label" for="id_receivers">{{form.receivers.label}}</label>
      {{form.receivers}}
    </div>

    <div class="form-group">
      <label class="h5 label" for="id_emails">{{form.emails.label}}</label>
      {{ form.emails }}
    </div>

    <div class="form-group row">
      <label class="h5 col-sm-3 col-form-label" for="id_title">{{form.title.label}}:</label>
      <div class="col-sm-9">
        {{ form.title }}
      </div>

    </div>

    <div class="form-group row">
      <label class="h5 col-sm-3 col-form-label" for="id_text">{{form.text.label}}:</label>
      <div class="col-sm-9">
        {{ form.text }}
      </div>
```

```
      </div>

      <div class="form-group row">
        <label class="h5 col-3 col-form-label" for="id_send_date">{{form.send_date.label}}:</label>
        <div class="col-9">
          {{form.send_date}}
        </div>
      </div>

      <button class="btn btn-primary" type="submit">Submit form</button>
      <input name="next" type="hidden" value="{{ next }}"/>
      {%if form.errors%}
        <div class="alert alert-danger" role="alert">
          {{form.errors}}
        </div>
      {%endif%}
    </form>

    <form class="mt-1" action="{% url 'just_delete' message.url %}" method="post">
      {% csrf_token %}
      <button type="submit" class="btn btn-danger">Delete</button>
    </form>


{% else %}
    <p>you are not authenticated</p>
    <a href="/login">Login</a>
{% endif %}

</div>
{% endblock %}
```

mysite/mymail/templates/mymail/login.html

```
{% include "mymail/base.html" %}
{% block content %}

<div class="container ml-2 mt-2 col-5">

{%if form.errors%}
    <div class="alert alert-danger" role="alert">
      {{form.errors}}
    </div>
{%endif%}

{% if user.is_authenticated %}
    <h1 class="ml-5 center text-center">You are already authenticated</h1>
{% else %}

<form method="post" action="{% url 'login_view' %}"> {% csrf_token %}
    <div class="form-group">
      <label for="Username">Username</label>
      <input name="username" class="form-control" id="Username"   placeholder="Enter username">
    </div>
```

```
  <div class="form-group">
    <label for="Password">Password</label>
    <input name="password" type="password" class="form-control" id="Password"
placeholder="Password">
    <small id="passwordHelp" class="form-text text-muted">We'll never share your password with anyone
else.</small>
  </div>

  <button type="submit" class="btn btn-primary">Login</button>
</form>

{% endif %}
</div>
{% endblock %}
```

mysite/mymail/templates/mymail/read_message.html

```
{% include "mymail/base.html" %}

{% block content %}

<div class="container ml-2 mt-2">

{% if message %}
    <p class="h4"> Message detail info</p>
    <hr>
    <p><strong>Sender: </strong>{{ message.sender }}</p>
    <p class="h8">
      {% if message.send_date %}
        {{ message.send_date }}
      {% else %}
        {{ message.pub_date }}
      {% endif %}
    </p>
    <br>

    <p class="h4"><strong>Receivers:</strong></p>
    <div class="ml-5">
      {% for receiver in message.receivers.all %}
        <p class="h6">{{ receiver.user.username }}</p>
      {% endfor %}
      {% for email in message.emails %}
        <p class="h6">{{ email }}</p>
      {% endfor %}
      <p class="h6">{{message.external_receivers}}</p>
    </div>
    <br>

    <p class="h3">{{ message.title }}</p>
    <br>
    <p class="h5 ml-5">{{ message.text }}</p>
    <br>

    <form action="{% url 'delete' message.url %}" method="post">
      {% csrf_token %}
    <input class="btn btn-danger" type="submit" value="Delete">
```

```
  </form>
{% endif %}

</div>
{% endblock %}
```

mysite/mymail/templates/mymail/registration.html

```
{% include "mymail/base.html" %}
{% block content %}

<div class="container ml-2 mt-2 col-5">

{%if form.errors%}
  <div class="alert alert-danger" role="alert">
    {{form.errors}}
  </div>
{%endif%}

{% if user.is_authenticated %}
  <h1 class="ml-5 center text-center">You are already authenticated</h1>
{% else %}

<form method="post" action=""> {% csrf_token %}
  <div class="form-group">
    <label for="Username">Username</label>
    <input name="username" class="form-control" id="Username"   placeholder="Enter username">
    <small id="loginHelp" class="form-text text-muted">Required. 150 characters or fewer. Letters, digits and
@/./+/-/_ only</small>

  </div>

  <div class="form-group">
    <label for="Email">Email address</label>
    <input name="email" type="email" class="form-control" id="Email" aria-describedby="emailHelp"
placeholder="Enter email">
  </div>

  <div class="form-group">
    <label for="Password">Password</label>
    <input name="password1" type="password" class="form-control" id="Password"
placeholder="Password">
    <small id="passwordHelp" class="form-text text-muted">We'll never share your password with anyone
else.</small>
    <small id="passwordHelp2" class="form-text text-muted">Your password can’t be too similar to your
other personal information.</small>
    <small id="passwordHelp3" class="form-text text-muted">Your password must contain at least 8
characters.</small>
    <small id="passwordHelp4" class="form-text text-muted">Your password can’t be a commonly used
password.</small>
    <small id="passwordHelp5" class="form-text text-muted">Your password can’t be entirely
numeric..</small>
  </div>

  <div class="form-group">
```

```
      <label for="Password2">Password confirmation:</label>
      <input name="password2" type="password" class="form-control" id="Password2"
placeholder="Password">
   </div>

   <button type="submit" class="btn btn-primary">Register</button>
   <input type="hidden" name="next" value="{{ next }}" />
</form>

{% endif %}
</div>
{% endblock %}
```

mysite/mymail/templates/mymail/search.html

```
{% include "mymail/base.html" %}

{% block content %}

<div class="container ml-2">
{% load mymail_extras %}

{% if user.is_authenticated %}
   <p class="h2 mt-2">{{what_search}}</p>
   {% if messages %}
      <ul class="list-group list-group-flush">
         {% for message in messages %}
            <li class="list-group-item">
               {% if what_search == "Templates" %}
                  <a style="display:block" href="{% url 'create_from_template' message.url %}" class="text-
decoration-none">
                     <div class="h4 text-primary">
                        <div>
                           {%if message.title%}
                              {{ message.title    }}
                           {%else%}
                              No title
                           {%endif%}
                        </div>
                        <div class="h6 text-secondary text-truncate">
                           {{message.text }}
                        </div>
                     </div>
                  </a>
               {% elif what_search == "Received messages" %}
                  {% if message.read %}
                     <a style="display:block" href="{% url 'read_message' message.message.url %}">
                        <div class="h4 text-secondary">
                           <div>
                              {%if message.message.title%}
                                 {{ message.message.title    }}
                              {%else%}
                                 No title
                              {%endif%}
                           </div>
```

```
                        <div class="h6 text-secondary text-truncate">
                          {{message.message.text}}
                        </div>
                      </div>
                    </a>
                  {% else %}
                    <a style="display:block" href="{% url 'read_message' message.message.url %}">
                      <div class="h4 text-primary">
                        <div>
                          {%if message.message.title%}
                          {{ message.message.title    }}
                          {%else%}
                          No title
                          {%endif%}
                        </div>
                        <div class="h6 text-secondary text-truncate">
                          {{message.message.text}}
                        </div>
                      </div>
                    </a>
                  {% endif %}
                {% else %}
                  {%if not message.send_date or  message.date < now  %}
                    <a style="display:block" href="{% url 'read_message' message.url %}">
                      <div class="h4 text-secondary">
                        <div>
                          {%if message.title%}
                            {{ message.title    }}
                          {%else%}
                            No title
                          {%endif%}
                        </div>
                        <div class="h6 text-secondary text-truncate">
                          {{message.text}}
                        </div>
                      </div>
                    </a>
                  {% else %}
                    <a style="display:block" href="{% url 'read_message' message.url %}">
                      <div class="h4 text-primary">
                        <div>
                          {%if message.title%}
                            {{ message.title    }}
                          {%else%}
                            No title
                          {%endif%}
                        </div>
                        <div class="h6 text-secondary text-truncate">
                          {{message.text}}
                        </div>
                      </div>
                    </a>
                  {% endif %}
                {% endif %}
              </li>
            {% endfor %}
```

```
        <nav aria-label="Page navigation example">
          <ul class="pagination">
            {% if  page|subtract:1 > -1%}
            <li class="page-item"><a class="page-link"
href="{{ page|subtract:1}}">{{ page|subtract:1}}</a></li>
            {% endif %}
            {% if more%}
            <li class="page-item"><a class="page-link" href="{{ page|add:1}}">{{ page|add:1}}</a></li>
            {% endif %}
          </ul>
        </nav>
      </ul>
    {% else %}
      <p>No messages.</p>
    {% endif %}
{% else %}
    <h1 class="ml-5 center text-center">You are not authenticated</h1>
{% endif %}
</div>

{% endblock %}
```

mysite/requirements.txt

```
APScheduler==3.6.3
asgiref==3.2.7
certifi==2020.4.5.1
chardet==3.0.4
confusable-homoglyphs==3.2.0
Django==3.0.6
django-extensions==2.2.9
django-multi-email-field==0.6.1
django-registration==3.1
idna==2.9
psycopg2==2.8.5
pydotplus==2.0.2
pyparsing==2.4.7
python-decouple==3.3
pytz==2020.1
requests==2.23.0
six==1.15.0
sqlparse==0.3.1
tzlocal==2.1
urllib3==1.25.9
```

mysite/.env

```
SECRET_KEY = your Django secret key
DB_NAME = your db name
DB_USER = your db user
DB_PASSWORD = your db password
DB_HOST=127.0.0.1
EMAIL_HOST = your email host
EMAIL_HOST_USER = your email login
EMAIL_HOST_PASSWORD = your email password
```